МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Методичні рекомендації до самостійної роботи з математичних дисциплін з використанням програмного середовища R

для студентів усіх спеціальностей першого (бакалаврського) рівня

Харків
ХНЕУ ім. С. Кузнеця
2017

УДК 519.85(07.034)
М54

Укладачі: Л. М. Малярець
О. Г. Тижненко

Затверджено на засіданні кафедри вищої математики і економіко-математичних методів.
Протокол № 7 від 15.03.2017 р.

Самостійне електронне текстове мережеве видання

Методичні рекомендації до самостійної роботи з математич-
М54 них дисциплін з використанням програмного середовища R для студентів усіх спеціальностей першого (бакалаврського) рівня [Електронний ресурс] / уклад. Л. М. Малярець, О. Г. Тижненко. – Харків : ХНЕУ ім. С. Кузнеця, 2017. – 85 с.

Подано основи з програмування у програмному середовищі R, а також приклади розв'язання задач з математичних дисциплін в R. Розв'язання задач супроводжується стислим викладом теоретичних положень, що допомагає студентам більш досконало засвоїти теоретичні знання та набути навичок їх самостійного застосування у розв'язанні практичних задач з математичних дисциплін за допомогою R.

Рекомендовано для студентів усіх спеціальностей першого (бакалаврського) рівня денної форми навчання.

УДК 519.85(07.034)

Вступ

Сучасний етап розвитку економіки характеризується збільшенням інтересу фахівців до вирішення проблем економічного аналізу за допомогою економіко-математичних методів та сучасних методів оброблення великих масивів інформації. Економіко-математичні методи та моделювання вже стало частиною економічного аналізу і відображає поєднання економічних, математичних та статистичних знань.

Економетрика, наприклад, займає одне з провідних місць у системі підготовки економістів на сучасному етапі. Спільно з іншими економічними та математичними дисциплінами економетрика формує нове економічне мислення у майбутніх фахівців, яке базується на сучасних методах опрацювання великих масивів економічної інформації.

У результаті вивчення нормативної дисципліни "Економетрика" студенти набувають компетентності технології складання економетричних моделей та оволодівають сучасними методами аналізу явищ і процесів в економіці за допомогою цих моделей; засвоюють базові поняття проблеми специфікації моделей; оволодівають методами перевірки виконання основних передумов застосовності методу найменших квадратів для визначення параметрів моделей та методами складання статистичних інференцій; оволодівають базовими методами аналізу даних.

Поруч із аудиторними заняттями з математичних дисциплін, важливою формою роботи студента є самостійний перегляд теоретичного матеріалу та виконання прикладів, що сприяють кращому засвоюванню теорії та придбанню практичних навичок роботи з даними.

Формування компетентностей фахівця у галузі економіки передбачає опанування сучасних методів опрацювання великих масивів даних. Відповідні обчислення, які при цьому виникають, на сучасному етапі розвитку світового суспільства здійснюють, в основному, у програмному середовищі R.

Розділ 1. Введення в R та RStudio

1.1. Початок роботи в RStudio

Використання R як програмного середовища в економічних дослідженнях передбачає використання програм, які написані мовою R. Дослідження можна безпосередньо виконувати в самої програмі R, але зручніше працювати в інтегрованому середовищі RStudio, яке навмисно створено для полегшення складання функцій користувача та маніпуляцій із файлами даних. Водночас користувач створює та виконує функції в RStudio та не звертається до R, але саме R виконує усі команди, які набираються у вікні виконання "Console" інтегрованого середовища RStudio. Тому для роботи в RStudio необхідно встановлювати на комп'ютер як RStudio так і R. Установку R слід проводити на сайті www.r-project.org. Установку RStudio слід проводити на сайті www.rstudio.com.

Для роботи в RStudio необхідно знати призначення функціональних кнопок панелі RStudio та основи мови R. Розглянемо основні моменти роботи в R за допомогою панелі RStudio, яка зображена на рис. 1.1.

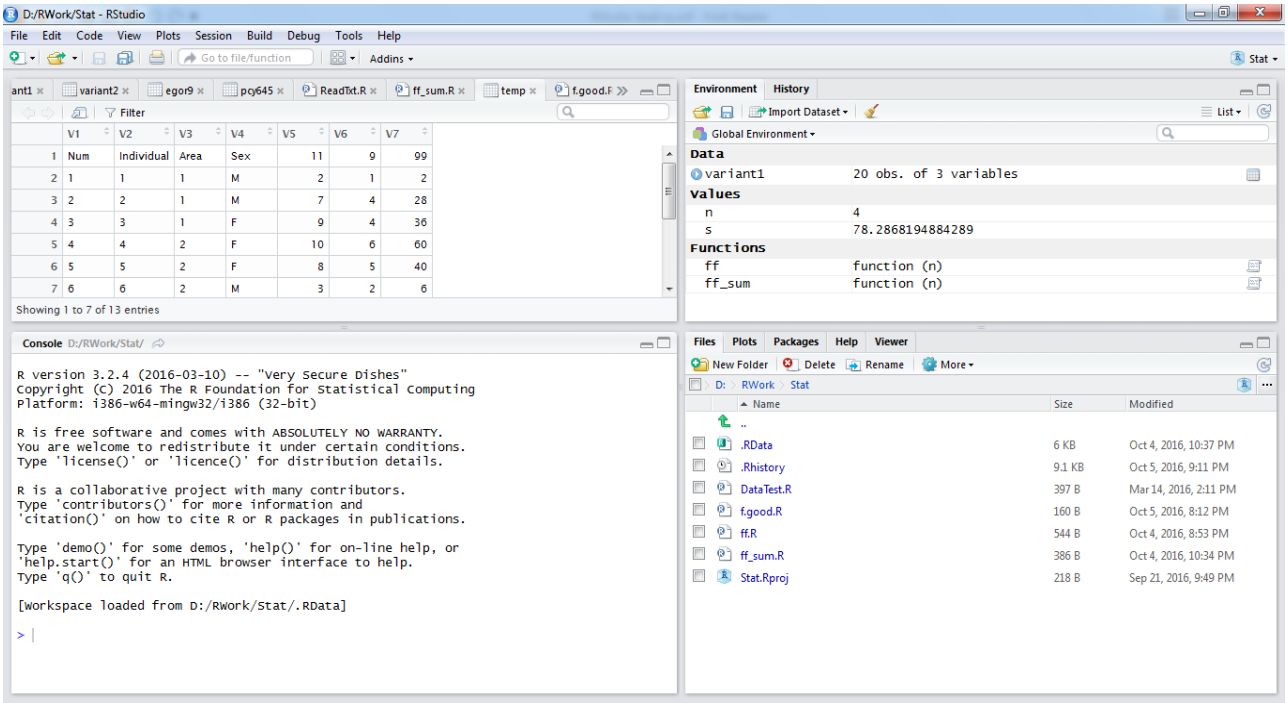


Рис. 1.1. **Панель RStudio**

Панель складається із чотирьох вікон: вікна Скриптів (редактор, верхній лівий квадрант), вікна Оточення (робоча область, верхній правий

квадрант), Консолі (вікно виконання, нижній лівий квадрант) та вікна Файлів (нижній правий квадрант).

Якщо робота відбувається на університетському комп'ютері, оновлення версій R та RStudio проводиться адміністратором локальної мережі. Версія R з'являється у вікні виконання (рис. 1.1) у першій стрічці автоматичного повідомлення, наприклад: R version 3.2.4 (2016-03-10) – та версія, яка буде розглянута у цій роботі. Версія RStudio може бути перевірена кліком кнопки "Help" головного меню і далі "About RStudio" (рис 1.2). При цьому з'явиться картинка, на якій можна побачити також пункт меню "R Help", який дозволяє потрапити до багатьох джерел, які дозволяють удосконалити роботу в R, а також до наукової літератури щодо застосування програмного середовища R в економічних, економетричних, соціальних та інших дослідженнях.

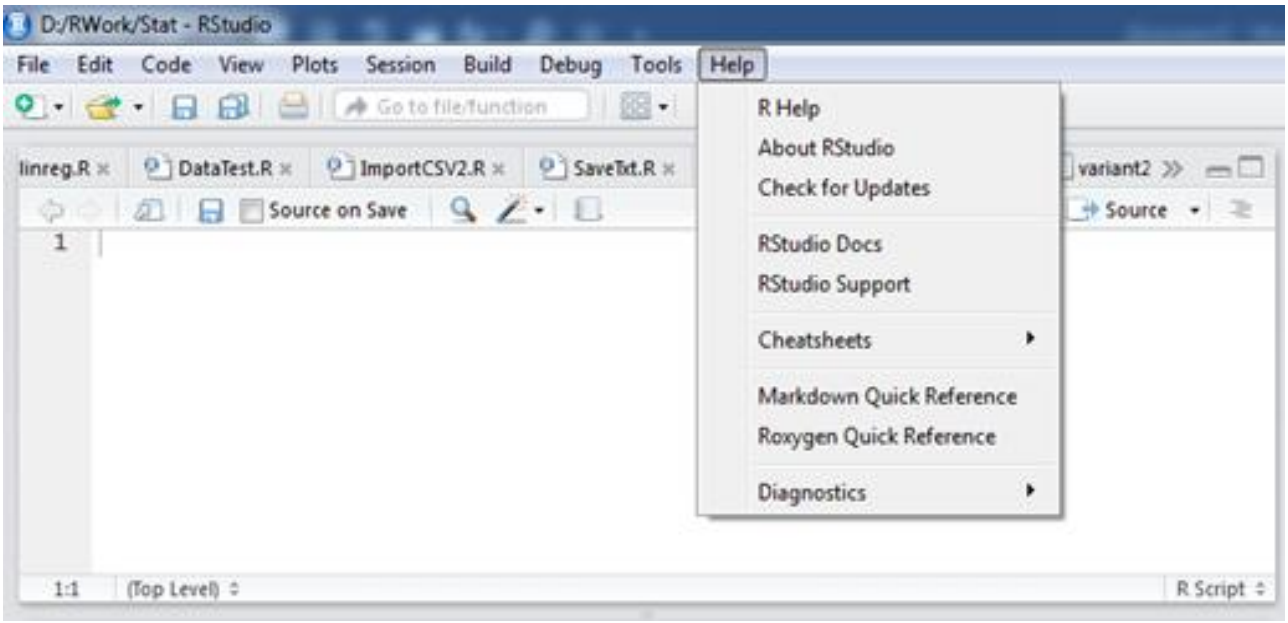


Рис. 1.2. **Вікно Скриптів**

У даній розробці розглядається версія RStudio 0.99.892 – © 2009-2016 RStudio, Inc. Номер версії висвічується після кліку на полі "About RStudio". Слід зауважити, що при установці обох програм на власний комп'ютер необхідно встановлювати останню версію програм, а потім періодично поновлювати програми самостійно, тому що автоматичного оновлення поки що немає. Якщо на комп'ютери встановлені не останні версії цих програм, може скластися ситуація, коли не можна бути встановленою деяка програма, яку ви знайшли в Інтернеті. При цьому у Консолі (Console) повідомляється (англійською мовою) про неможливість

встановити програму, або пакет, але не пояснюється чому. Слід знати, що причина саме та, про яку йшла мова.

Із попередньої інформації зрозуміло, що вікно "Консоль" призначено для запуску програм та спілкування з програмним середовищем R. Більш детально це вікно зображено на рис. 1.3. У цьому вікні міститься важлива інформація, без знання якої неможливо запустити жодну програму. У самому верхньому рядку вікна написано: "Console D:/RWork/Stat/". Це вказано у робочій директорії, яка оголошується користувачем. Якщо R та RStudio встановлюються на комп'ютері уперше, робочий директорій треба зробити вручну. Для цього на диску D, наприклад, слід зробити папку. У даному разі зроблена папка D:\RWork, у якій будуть зберігатися усі програми користувача, які скомпоновані в проекти (Projects), тобто по суті папки в D:\RWork. Якщо ми хочемо працювати з проблемами статистики і для цього відкрили проект Stat у робочій директорії, то ми повинні у Консолі вказати *путь до функцій* цього директорія. Для цього в R існує функція **source()**. Тобто слід виконати команду: **source**('D:/RWork/Stat/імя.функції.R').

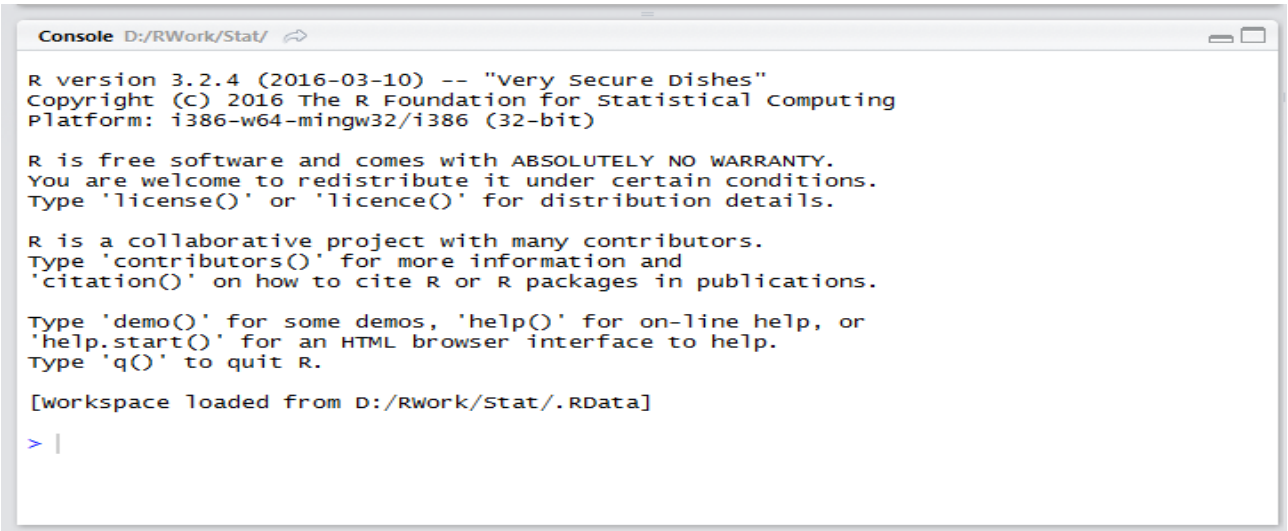


Рис. 1.3. **Вікно Консоль**

Вікно Консоль (Console), або вікно виконання, призначене для виконання програм та безпосередніх обчислень.

У вікні Консоль вказана також можливість цитувати створені програми у літературі, **citation()**, звертатися до демо-файлів, **demo()**, та файлів помочі, **help()** та **help.start()**.

Слід зазначити, що нові проекти (папки) відкриваються зсередини RStudio у вікні файлів за допомогою кнопки"Files"\ "New Folder" (рис. 1.4).

На рис. 1.4 показаний випадок, коли в робочої директорії у відкрито проект Stat, що відображено програмою RStudio як "Stat.Proj" у нижньому рядку папки Stat, а також папки: "Regress" та "Temp", які належать до робочої директорії D:\RWork, тобто використовуються при роботі безпосередньо у цьому директорії. Однак перш ніж це можна зробити, ми повинні повідомити R, яку директорію ми вибрали робочою. Для цього в R існує програма **setwd()**, яку треба виконати у Консолі.

Якщо ми хочемо вибрати в якості робочої директорії D:\RWork, то необхідно набрати у Консолі (рис. 1.3):

setwd('D:/RWork') (1.1)

та натиснути на Enter. Водночас слід звернути увагу на те, що в R путь задається не так, як прийнято у системному програмуванні: "\". Тобто замість оберненого слешу використовується прямий слеш: "/".

До установлення робочої директорії у вікні Файлів було пусто. Після виконання команди (1.1) у вікні з'являються перші два рядка, які свідчать про створення двох вбудованих папок: .RData та .Rhistory.

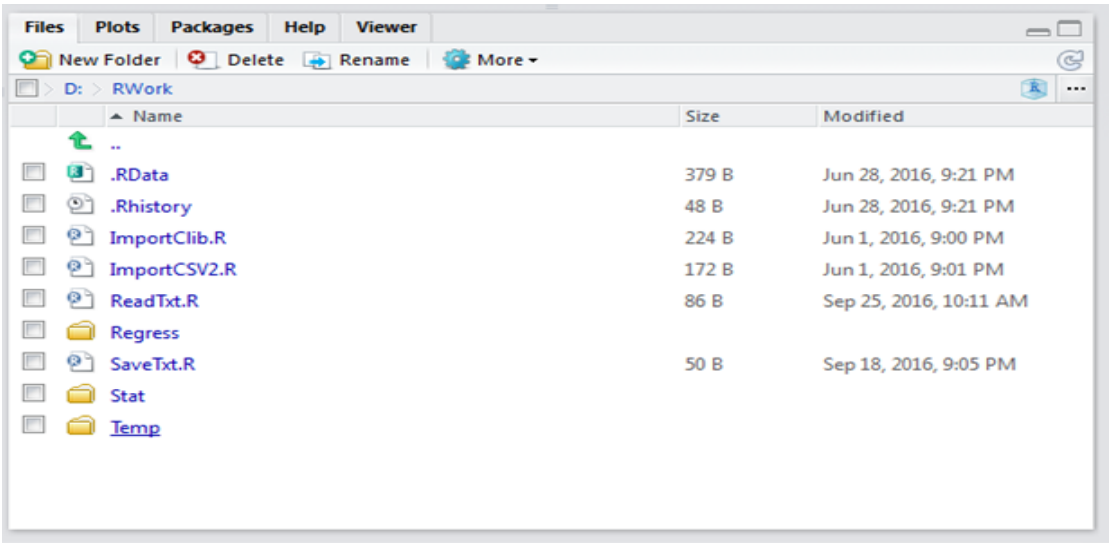


Рис. 1.4. **Вікно файлів у директорії D:/RWork**

Якщо цього не відбувається, слід натиснути кнопку "More" та вибрати "Go to the work directory".

Якщо робота відбувається безпосередньо у робочій директорії D:/RWork, то програми користувача, які створені у вікні Скриптів, записуються безпосередньо у D:/RWork. У файлі RData записується зміст електронного оточення: програми, які виконувалися, змінні, які створювалися та дані, які використовувалися. У Rhistory автоматично записуються усі команди, які виконувалися.

Після установлення робочої директорії можна працювати безпосередньо у директорії D:/RWork. На рис. 1.4 видно, що у робочій директорії вже створені декілька функцій: **ImportClib.R**; **ImportCSV2.R**; **ReadTxt.R**; **SaveTxt.R**. Крім того, створені три папки: Stat, Regress та Temp. Слід звернути увагу на те, що папка Stat виглядає однаково з папками Regress та Temp, але це зовсім інша папка – це є Проект "Stat", тобто Stat Project. На відміну від папок Regress та Temp ця папка містить у перших двох рядках вбудовані директорії .RData та .Rhistory, як можна бачити на рис. 1.5.

Зауважимо, що нові папки у робочої директорії або у Проекті створюються у вікні Файлів за допомогою кнопки "New Folder" у другому рядку панелі вікна. Нові папки дозволяють сортувати програми за змістом досліджень.

Функції створюються у вікні Скриптів (див. рис. 1.2), у який вбудовано редактор. Крім вікна Файлів, проекти створюються також у вікні Скриптів: File>New Project. У проектах зібрані програми, які стосуються певних досліджень.

На рис. 1.5 зображена ситуація у вікні Файлів під час роботи у директорії D:/RWork/Stat. Про це свідчить перший рядок вікна D: > RWork>Stat.

Якщо переходу не відбулося, слід натиснути кнопку "More" та вибрати "Go to the work directory".

Видно, що у цієї директорії також є вбудовані директорії .RData та .Rhistory, а також створені функції: **DataTest.R**; **f.good.R**; **ff.R**; та **ff_sum.R**, ще фігурує назва вже існуючого проекту: Stat.Proj. Назви функцій показують допустимі варіанти ідентифікацій нових функцій під час створення.

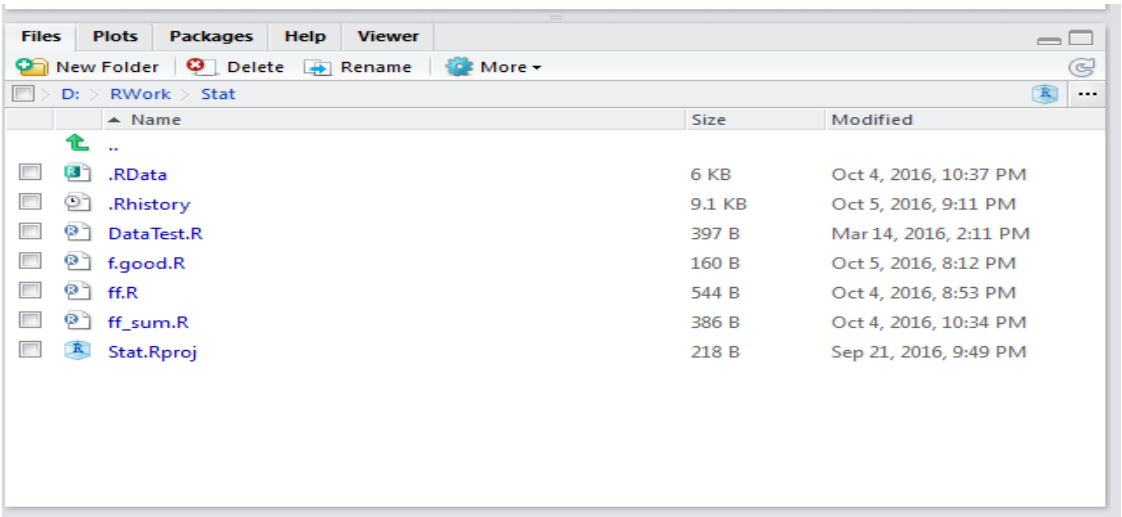


Рис. 1.5. **Вікно файлів у директорії D:/RWork/Stat**

Якщо клікнути на функцію у цьому вікні, – функція з'явиться у вікні Скриптів. Наприклад, клікаємо на функцію ff.sum.R. У вікні Скриптів з'явиться програма (рис. 1.6).

Редактор RStudio дозволяє модифікувати програму, яка вже існує у робочій директорії, після її виклику, або написати нову програму з новою назвою.

Розглянемо основні моменти роботи у редакторі, які дозволяють використовувати функції, які вже існують у робочої директорії, а також створювати нові функції та зберігати їх як у робочої директорії, так і в інших директоріях.

Програмне середовище R створене таким чином, що дозволяє використовувати як функції користувача, так і функції пакетів, які необхідно встановлювати з офіційного сайту CRAN: https://cran.r-project.org/web/packages/available_packages_by_name.html. Цей сайт можна знайти також у браузері, якщо набрати: "available cran packages by name". Можна також використати сайт: https://cran.r-project.org/web/views/, де пакети згруповані за темами.

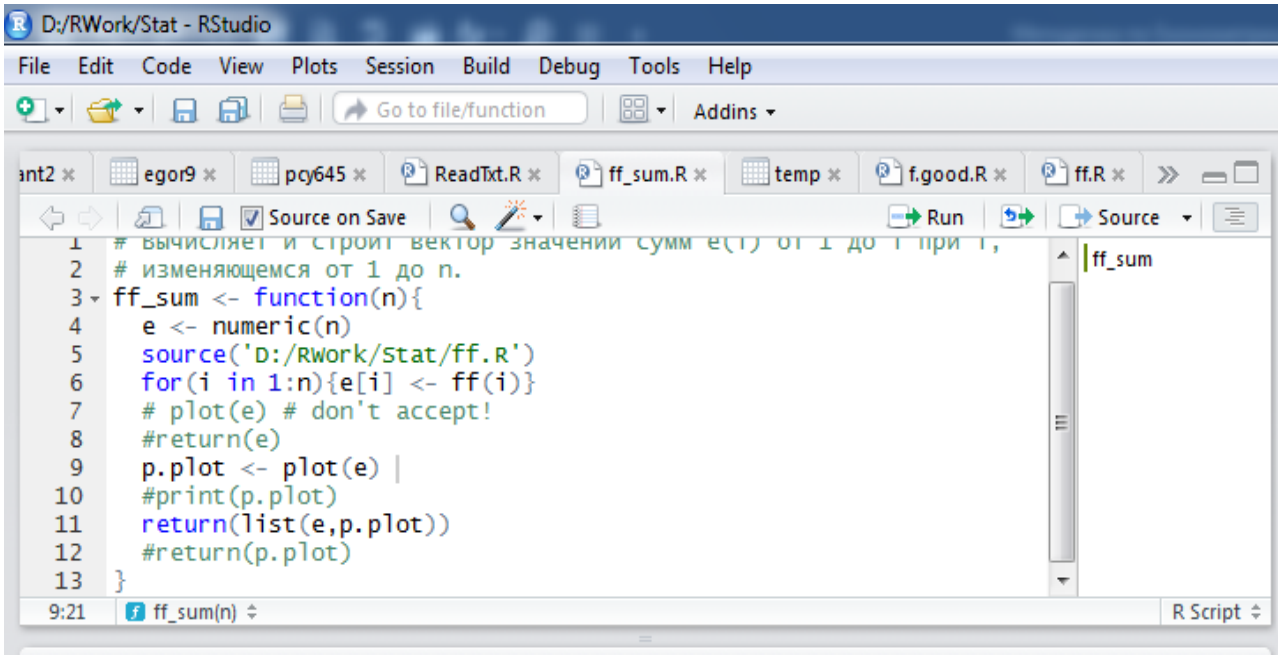


Рис. 1.6. **Редактор**

Показана функція, яка викликана з вікна Файлів, або створена у самому редакторі, як нова.

Якщо ви знайшли потрібній пакет, необхідно запам'ятати його назву, зверніться до вікна Файлів, натисніть кнопку "Packages" головного меню. У цьому разі зміниться меню задач (другий рядок меню): з'явиться кнопка "Install", яку треба натиснути. До того ж з'явиться вікно (рис. 1.7).

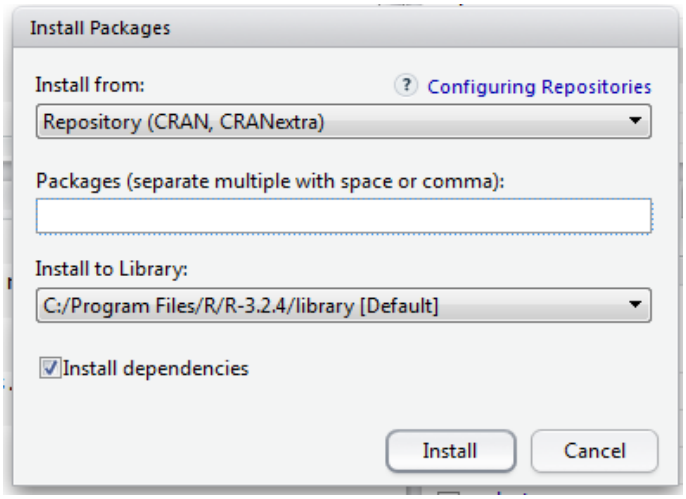


Рис. 1.7. **Установка пакету**

У пусте поле слід вписати назву пакету англійською мовою та натиснути кнопку "Install". Після цього у списку пакетів з'явиться назва встановленого пакету. Якщо клікнути на назву пакету, викликається "Help", де приведена повна документація до пакету та усіх функцій, які він містить.

Слід зауважити, що функції пакету за структурою не відрізняються від функцій користувача, але мають привілей – під час їх використання не потрібно вказувати шлях до функції, у той час як під час використання власних функцій це обов'язково. Для використання власних функцій вони повинні бути завантажені в електронне оточення (Global Environment), яке відображено у вікні Оточення (рис. 1.1). Це виконується, коли ми виконуємо команду **source**('шлях до функції') у Консолі, або використовуємо "Source on Save" у редакторі.

Наведені особливості використання власних функцій проілюстровано на рис. 1.8.

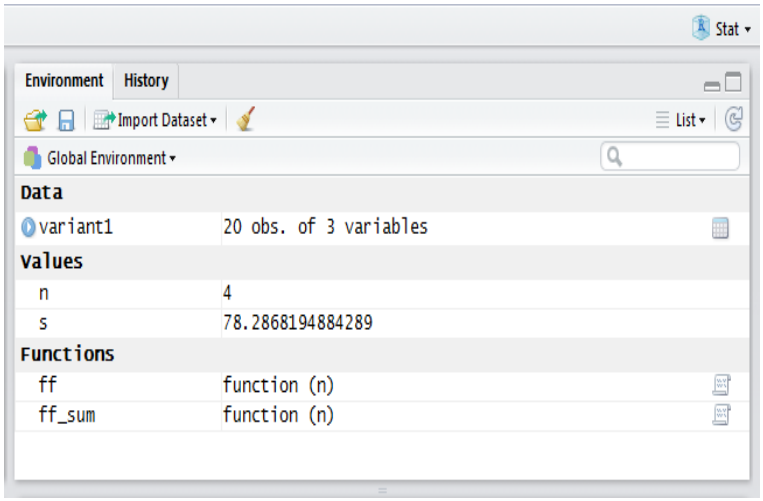


Рис. 1.8. **Глобальне оточення**

Для того, щоб використати функцію **ff_sum.R**, яка показана у редакторі на рис. 1.6, вона повинна бути завантажена у глобальне оточення, про що свідчить її поява у вікні Оточення (рис. 1.8).

Дуже важливо замітити, що функція **ff_sum.R** звертається до іншої власної функції **ff.R**. Ця функція теж повинна бути у глобальному оточенні. Вона там і є, якщо подивитися на рис. 1.8. Але це досягається тим, що у тілі функції **ff_sum.R** є команда: **source**('D:/RWork/Stat/ff.R'), як можна бачити на рис. 1.6. Ця команда як раз і задає шлях до функції **ff.R**.

Детальніше структура функції та її складання будуть розглянуті далі. Зараз відмітимо тільки, що знак номеру # в англійській розкладці, який можна замітити в програмі на рис. 1.6, означає коментар. Рядок, який починається із цього знаку, програма R не виконує. Це дозволяє робити пояснення в тілі функції.

Якщо функція **ff_sum.R**, наприклад, створена та записана у робочій директорії, то це ще не означає, що вона може бути виконана у Консолі, якщо її не завантажили у Глобальне Оточення за допомогою команди **source**('D:/RWork/Stat/ff_sum.R') або з редактору за допомогою зберігання з відміткою у вікні "Source on Save".

Видалимо, наприклад, функцію **ff_sum.R** з Глобального Оточення за допомогою команди **rm()** та спробуємо виконати цю функцію у Консолі. Ми отримуємо:

```
> rm(ff_sum)
> e <- ff_sum(8)
Error: could not find function "ff_sum" .
```

Тут перша команда видаляє функцію **ff_sum.R** з Глобального Оточення після натиснення Вводу. Друга команда виповнює функцію **ff_sum.R** для значення *n* = 8. По суті, це означає $n = 8$, тільки слід пам'ятати, що знак рівності "=" в R, як правило, замінюється на знак "<-". Для цього в R передбачена гаряча клавіша: Alt + "–".

Спроба виконати функцію, яка не завантажена у Глобальне Оточення приводить до помилки та пояснення: "нема можливості знайти функцію "ff_sum"".

Після того, як функція **ff_sum.R** буде завантажена у Глобальне Оточення, її можна виконувати:

```
source('D:/RWork/Stat/ff_sum.R')
> e <- ff_sum(8)
> e
[[1]] # denoting a list
[1] 1.2629543 0.9367209 2.2665202 3.5389495 3.9535909 2.4136409 1.4850739
[8] 1.1903534

[[2]]
NULL .
```

Тут перший рядок задає шлях до функції та завантажує її у Глобальне Оточення (після натиснення Вводу). Другий рядок – команда виконання функції $e = ff\_sum(n)$ при $n = 8$.

Зауважимо, що отримані значення вектора *e* мають нечисловий формат, оскільки виводилися з функції у виді списку (list). Тобто ми не можемо проводити із цими числами арифметичних операцій. Для перетворення даних у числовий формат слід застосувати функцію **unlist()**:

```
w <- unlist(e)
> w
[1] 1.2629543 0.9367209 2.2665202 3.5389495 3.9535909 2.4136409 1.4850739
[8] 1.1903534
> w[1]
[1] 1.262954 .
```

Тобто ми замінили вектор-список *e* на числовий вектор *w* та знайшли його першу координату *w*[1]= 1.262954. Слід замітити, що координати числових векторів в R позначаються за допомогою квадратних дужок: *w*[i].

Усі розрахунки, які проводяться у Консолі, відображаються у Глобальному Оточенні. Наприклад, результатом розрахунків які пов'язані з функцією **ff_sum.R** є вектор e у символьних змінних у вигляді списку (list):

```
> e
[[1]]
[1] 1.2629543 0.9367209 2.2665202 3.5389495 3.9535909 2.4136409 1.4850739
[8] 1.1903534

[[2]]
NULL .
```

Ця змінна відображається у Глобальному Оточенні у першому рядку розділу "**Values**", рис. 1.9. Також відображається числовий вектор *w.*

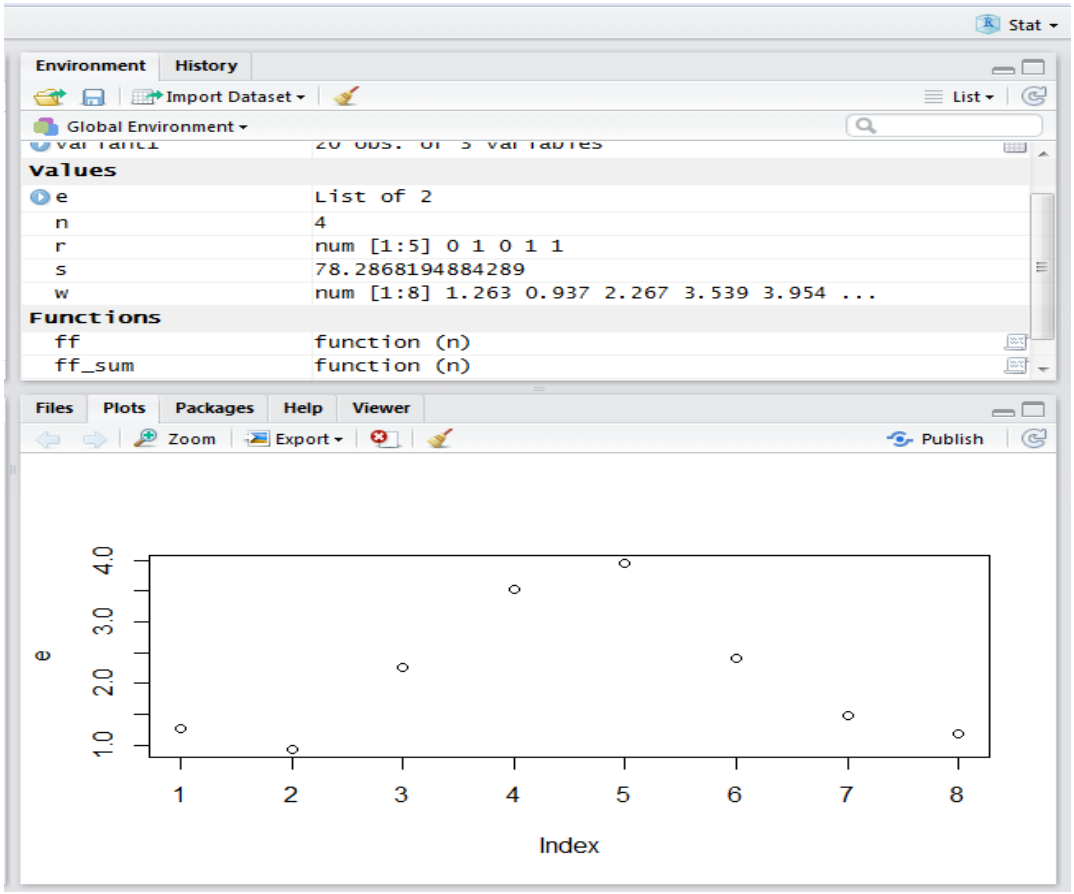


Рис. 1.9. **Глобальне Оточення та вікно Файлів**

У вікні Файлів відкрито вікно **Plots** у результаті виконання функції **ff_sum.R**, у якому є побудованим графік вектора e від індексу.

Розглянемо, що відбувається у цих функціях. У програмі **ff_sum.R** підсумовуються послідовні значення генератора випадкових чисел однієї реалізації, які задаються у функції **ff.R** для нормального та центрованого рівномірного ЗР в залежності від розміру вибірки. Бачимо, що ця сума відчуває сильні макро-коливання, яких не повинно бути! Тобто спостерігається явна корельованість псевдо-випадкових чисел!

Як видно з розділу "Functions", завантажені також функції, які використовувались: **ff.R** та **ff_sum.R**. Усі змінні константи та функції, які завантажені у процесі роботи в R у поточної сесії можуть бути збережені для наступних сесій, а можна убрати в разі закінчення роботи. Тобто R пропонує вибір по закінченні роботи (рис. 1.10).

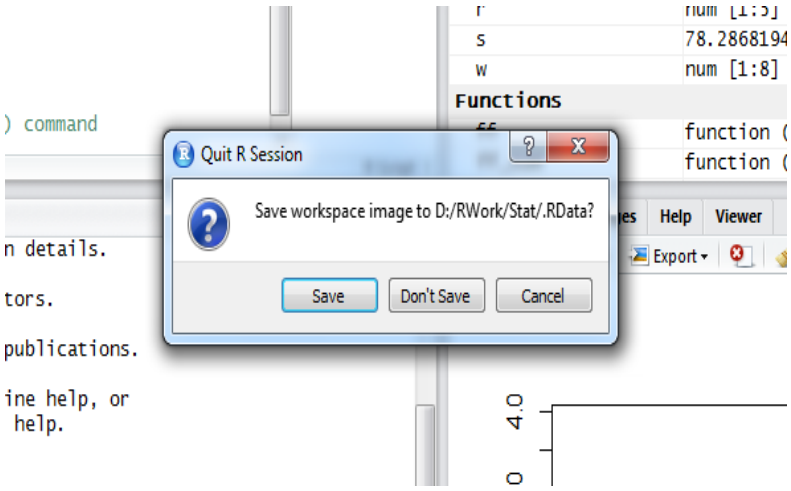


Рис. 1.10. **Вихід з R**

Дуже важливо залишати тільки те, що дійсно потрібно для наступної роботи, тому що зайві змінні перевантажують пам'ять комп'ютера. Для того, щоб не залишати не потрібні змінні, їх треба видаляти до виходу із сесії за допомогою функції **rm()**, яка виконується у Консолі. Наприклад, **rm(n)** видаляє константу *n*, яка дорівнює 4, рис. 1.9. Також можуть бути не потрібними вектор *r* та константа *s*, які убираються командами **rm**(r) та **rm**(s). Після видалення непотрібних змінних, можна виходити з R натиснув поле з хрестиком. Після цього з'явиться вікно "Quit Session", рис 1.10, на котрому слід натиснути кнопку "Save". Є також інший варіант виходу з R. Якщо вважається непотрібним залишати змінні та функції, які використалися у сесії, у Глобальному Оточенні, слід натиснути у вікні "Quit Session" (рис 1.10) кнопку "Don't Save". Тоді у наступній сесії прийдеться знову завантажувати Глобальне Оточення змінним та функціями, якщо вони виявляться потрібними.

Останнє зауваження до загальних питань щодо початку роботи в R, стосується позначень функцій та файлів даних у редакторі R. Функції позначаються значком документа з літерою R. Що стосується даних, то в R дані, в основному зберігаються у файлах "**data.frame**", які являють собою таблицю з назвами змінних, що показано на рис. 1.11. У редакторі така таблиця позначається значком, схожим на табличку. На рис. 1.11 показано результат натиснення на кнопку "temp". При цьому "temp" зі значком таблиці висвітлюється, показуючи, що таблиця, яка з'явилася у редакторі, і є файл даних (**data.frame**) позначений назвою "temp". Як видно з рис. 1.9, цей файл даних не завантажений у Глобальне Оточення у даної сесії. Це означає, що не можемо звернутися до цих даних.

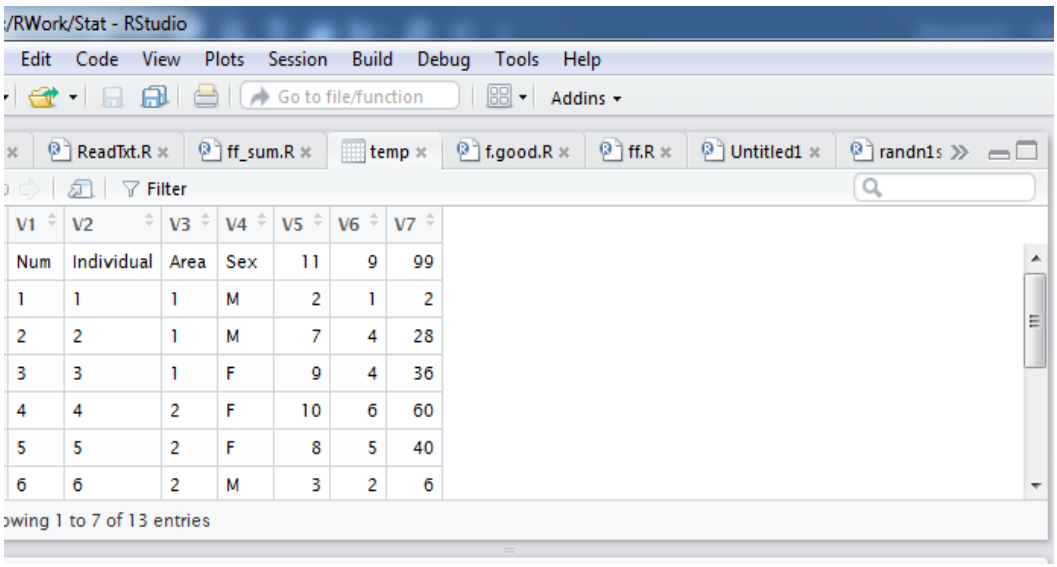


Рис. 1.11. **Назва змінних у редакторі**

У даному прикладі файл даних "temp" є звичайний текстовий файл, temp.txt, який зберігається у окремій папці D:\RTxt. Слід тільки мати на увазі, що десяткові роздільники повинні бути крапками. Якщо ні, то це слід враховувати на програмному рівні.

Зауважимо, що економічні дані, як правило зберігаються або у файлах Excel, або у текстових файлах, або у файлах .CSV2. Має сенс зберігати їх у різних папках. Проблемам обробки даних у різних форматах присвячений у даному посібнику окремий розгляд.

Якщо, як у даному прикладі, дані зберігаються у текстовому файлі, для їх використання вони повинні бути прочитані програмою R: **read.table.R**. При багаторазовому використанні різних даних зручно створити собі у редакторі окрему функцію читання текстових файлів:

```
temp <- read.table("D:/RTxt/temp.txt",
                   header = TRUE,sep="") ,
```

та зберегти її у робочий директорій під назвою **Read.Text.R**, наприклад. Ця функція викликається з робочої директорії у редактор простим кліком з вікна Файлів. У редакторі весь текст функції виділяється курсором та натискується кнопка **Run** на панелі. При цьому у Консолі повинно з'явитися:

```
> temp <- read.table("D:/RTxt/temp.txt",
+                    header = TRUE,sep="") .
```

Одночасно у Глобальному Оточенні з'явиться запис у розділі **Data**: "temp 12 obs. of 6 variables". Це означає, що програма Read.Text.R привласнила таблиці даних назву "temp" (в R) та ця таблиця складається з 6 змінних (variables) розміром у 12 спостережень (obs. – observables). Після цих операцій дані готові до оброблення в R.

Слід звернути увагу на структуру та зміст функції Read.Text.R. По-перше, назва таблиці (temp) в R може бути будь-якої; по друге, у дужках функції **read.table()** указаний шлях до текстового файлу в лапках; далі через кому записано: header = TRUE, що означає, що текстовий файл містить заголовки змінних (стовпчиків даних); далі записано: sep="". Це означає, що у даних текстового файлу використані крапки у якості десяткових роздільниів, а не коми. Якщо заголовків нема, пишемо: header = FALSE замість header = TRUE. Якщо у даних текстового файлу використані коми у якості десяткових роздільників, пишімо: sep="," замість sep="".

Робота з даними більш детально розглядається далі. Що стосується графіка, який отриманий у вікні "Plots" на рис. 1.9, то побудова графіків та їх редакція також буде розглянута окремо.

1.2. Зручні функції загального характеру

Починаючи роботу в R, перш за все необхідно перевірити, який робочий директорій встановлено у даній сесії за допомогою команди **getwd()**:

```
> getwd()
[1] "D:/RWork/Stat" .
```

У цьому випадку R відповідає, що робочим директорієм є проект "Stat", який знаходиться у директорії D:/RWork .

Якщо робочої директорії не встановлено, R відповість на цю команду:

```
 > getwd()
[1] character(0) .
```

У цьому випадку необхідно встановити робочу директорію за допомогою команди **setdw()**, яку треба виконати у Консолі:

setwd("D:/RWork") ,

або відразу:

setwd("D:/RWork/Stat"),

якщо є намір працювати у конкретному проекті "Stat", а не у загальній директорії D:/RWork.

Наприклад, необхідно створити програму з назвою **normrnd()**. Необхідно перевірити, чи нема вже такої програми в робочій директорії. Виконуємо у Консолі команду:

```
> normrnd
Error: object 'normrnd' not found .
```

Тобто набираємо у командному рядку назву функції та натискуємо "Enter". Якщо такої функції нема у робочій директорії, R відповідає повідомленням про помилку, з якого зрозуміло, що R не може знайти таку функцію. У цьому випадку нову функцію можна називати цим ім'ям.

Під час пошуку потрібних програм корисною є програма **apropos**("ім'я функції, або частки функції"). Наприклад, ми шукаємо програму, яка пов'язана з нормальним розподілом та містить сполучення "norm". Виконуємо команду:

```
> apropos("norm")
 [1] ".rs.normalizeKeyboardShortcut"    ".rs.normalizePath"
 [3] ".rs.validateAndNormalizeEncoding" "dlnorm"
 [5] "dnorm"                            "norm"
 [7] "normalizePath"                        "plnorm"
 [9] "pnorm"                            "qlnorm"
[11] "qnorm"                             "qqnorm"
[13] "rlnorm"                           "rnorm" ,
```

та отримуємо список усіх функцій, які знаходяться у робочому директорії (функції користувача) і у всіх встановлених пакетах програм (Packages). Слід зауважити, що пошук функцій користувача із заданим словосполученням обмежений тільки робочою директорією. Вбудовані функції, у той же час, відшукуються у всіх встановлених пакетах.

Для того, щоб знайти, у якому пакеті знаходиться функція, зручно використати функцію **find**("ім'я функції"):

```
> find("dnorm")
[1] "package:stats" .
```

Щоб отримати довідку за змістом функції, необхідно набрати:

> ?dnorm та натиснути "Enter". Після цього, довідка завантажується у вікні Файлів (рис. 1.1).

Для того, щоб прочитати існуючу функцію користувача, необхідно просто набрати у Консолі її назву (без лапок). Наприклад:

```
> ff
function(n) {set.seed(0)
#e <- 0.5-runif(n)
e <- rnorm(n)
sum(e)} .
```

Для того, щоб прочитати існуючу вбудовану функцію теж можна використати той самий метод, але програми не усіх функцій пакетів можна продивитися таким чином. Наприклад, функція **lm()** розрахунку параметрів лінійної регресії, та усіх статистик, які із цим пов'язані, та дозволяє ознайомитися з її кодом. У той самий час функція транспонування **t()** цього зробити не дозволяє. Якщо виконати у Консолі команду:

```
> t
function (x)
UseMethod("t")
<bytecode: 0x075cf0a0>
<environment: namespace:base> ,
```

то отримуємо тільки вказівки, як це можна зробити.

Для статистичних функцій, подібних **lm**(), корисною є функція перегляду аргументів функції, **args()**. Наприклад,

```
> args(lm)
function (formula, data, subset, weights, na.action, method = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
    contrasts = NULL, offset, ...)
NULL .
```

Загальноприйняті позначення в R: Inf, NaN, NA and NULL, мають заступний зміст. Inf – нескінченність; NaN – не число; NA – пропущене або невизначене значення логічного типу; NULL – те саме, що NA у загальному випадку, або має логічний зміст: "поняття не визначено". Саме про це йдеться в останньому прикладі.

1.3. Виконання арифметичних та алгебраїчних операцій в R

Усі обчислення в R відбуваються у командному вікні (Консолі) за допомогою команд, які записуються після запрошення ">" при наявності миготливого курсору "|". Якщо команда не записується, слід активізувати вікно кліком мишкою.

Команди виконуються після натиснення клавіши Enter. У командному рядку можна писати декілька команд, але їх треба розділяти знаком ";".

Команди можна переносити також на другий рядок, якщо немає місця на першому рядку. Для цього необхідно натиснути клавішу Enter. При цьому на другому рядку у якості запрошення з'явиться знак "+", після котрого можна писати команди далі.

За допомогою клавіш "↑" та "↓" можливо вибирати попередні команди, що прискорює набір команд.

Для найменування команд, функцій та змінних слід скористатися англійською розкладкою клавіатури. Слід також пам'ятати, що R розрізняє великі та малі літери. Тобто "Temp" і "temp" – це різні назви.

У якості оператора присвоювання як правило застосовують знак "<–", для котрого в R можна застосовувати сполучення: "Alt + "–" ". Можна також використовувати звичайний знак "=" і це не приводить до помилки практично у всіх випадках.

У якості ідентифікаторів в R не можна використовувати літери: c та t. Літера c використовується для завдання вектора, тобто це функція **c()**, а літера t використовується для транспонування матриць, тобто це теж функція: **t()**.

Для виконання команд в R необхідно набрати команди латиною з урахуванням синтаксису мови та натиснути "Enter".

Арифметичні та алгебраїчні операції.

1. Додавання:

```
> 2+3
[1] 5

> a <- 2;b <- 3; z <- a+b
> z
[1] 5 .
```

Слід мати на увазі що R не виводить автоматично на печать (тобто на екран) значення ідентифікатора z під час обчислення за командою

z <- a+b . Для виведення результату на екран (на друк) необхідно набрати у наступному рядку z та натиснути Enter. Для автоматичного виведення результату на екран не слід писати ідентифікатор z:

```
> a <- 2;b <- 3; a+b
[1] 5 .
```

У цьому разі відразу з'являється результат у вигляді вектора з однією компонентою: "[1] 5". Однак слід враховувати, що подальше використання кінцевого результату неможливо, якщо результату не присвоєно ідентифікатор, як у попередньому прикладі: z <- a+b.

2. Множення:

```
> 2*(-3)
[1] -6

> a <- 2; b <- -3
> (z <- 2*a*b)
[1] -12
> z
[1] -12 .
```

У цьому прикладі обчислюється величина $z = 2ab$ за заданими величинах *a* та *b*. Показано варіант виведення результату безпосередньо: (z <- 2*a*b) та без цієї команди за допомогою натиснення z ще раз.

3. Ділення:

```
2/(-3)
[1] -0.6666667

> a <- 2; b <- -3
> z <- a/b
> z
[1] -0.6666667 .
```

Зауважимо, що ці чотири арифметичні дії можуть також бути виконані за допомогою зворотних лапок наступним чином:

```
> `+`(2,3)
[1] 5
> `-`(2,3)
[1] -1
> `/`(2,3)
[1] 0.6666667
> `*`(2,3)
[1] 6 .
```

Такий метод завдання арифметичних функцій може бути корисним під час обробки стовпчиків та рядів великих масивів економічних даних.

4. Зведення в ступень ($z = a^b$):

```
> a <- 2; b <- -3
> z <- a^b
> z
[1] 0.125 .
```

Якщо основою ступеня є число Непера (*e*), то для зведення у ступень ($z = e^b$) в R існує спеціальне позначення: ($z = \exp(b)$). Наприклад,

```
> b <- -2
> z <- exp(b)
> z
[1] 0.1353353 .
```

Або, для більш складної функції Гауса ($z = \exp(-x^2 / 2)$), наприклад,

```
x <- 2
> z <- exp(-x^2/2)
> z
[1] 0.1353353 .
```

Зауважимо, що в R вираз $x^2 / 2$ слід записувати саме як x^2/2 , а ні як (x^2)/2, у зв'язку з тим, що в R спершу виконується операція зведення у ступень, а потім арифметичні операції.

5. Обчислення логарифму.

$z = \lg(a)$.

```
> a <- 10
> z <- log10(a)
> z
[1] 1
```

$z = \ln(a)$.

```
> a <- exp(1)
> z <- log(a)
> z
[1] 1
```

$z = \log_2 a$

```
> a <- 2
> z <- log2(a)
> z
[1] 1 .
```

Тобто в R десятинний логарифм позначається $\log 10$, натуральний логарифм позначається $\log$, логарифм за основою 2 позначається $\log 2$. Для обчислення логарифмів за іншими основами необхідно використати формули перетворення від основи a до основи b:

$$\log_a N = \frac{\log_b N}{\log_b a}.$$

Якщо, наприклад, необхідно обчислити $\log_3 10$, то можна зробити попередні перетворення до десятинних логарифмів:

$$\log_3 10 = \frac{\lg 10}{\lg 3} = \frac{1}{\lg 3},$$

а потім провести обчислення в R:

```
N <- 10
> z <- 1/log10(3)
> z
[1] 2.095903 .
```

Можна також перевірити правильність обчислень за допомогою означення: $\log_3 10$ є ступень, у який слід возвести основу (3), щоб отримати число (10). Тобто $3^{2.095903}$ повинно дорівнювати 10. Перевіряємо в R:

```
> 3^z
[1] 10 .
```

Тобто обчислення зроблені правильно.

Слід зауважити, що в R не використовується звичайне позначення "e" для основи натуральних логарифмів (число Непера). Це число слід записувати в R як:

```
> e <- exp(1)
> e
[1] 2.718282 .
```

Далі, *у тілі однієї програми* ідентифікатор "e" буде сприйматися як число Непера. В іншій програмі це число необхідно ідентифікувати знову.

6. Обчислення тригонометричних функцій.

Функція синуса в R записується як **sin()**. Синус кута α записується в R як $\sin(alfa)$. Тобто необхідно ввести ідентифікатор кута α, наприклад, $alfa$, оскільки у мовах програмування усі команди та змінні записуються латиною. Слід також враховувати, що кути вимірюються у радіанах, замість звичних градусів. Це означає, що для обчислення $\sin(30^\circ)$, який дорівнює 0.5, наприклад, в R треба обчислювати функцію: $\sin(30*\pi/180)$, з урахуванням того, що в R замість π використовується сполучення *pi*. Тобто:

```
> sin(30*pi/180)
[1] 0.5 .
```

Функція косинуса в R записується як **cos()**. Наприклад, $\cos(60^\circ)$, який дорівнює 0.5 обчислюється в R наступним чином:

```
> cos(60*pi/180)
[1] 0.5 .
```

Інші тригонометричні функції в R мають нетрадиційні позначення.

Функція тангенса в R записується як **tan()**. Наприклад, $tg45^\circ$, який дорівнює одиниці, обчислюється в R наступним чином:

```
> tan(45*pi/180)
[1] 1 .
```

Функція котангенса в R відсутня. Наприклад, $ctg45^\circ$, який теж дорівнює одиниці, може обчислюватися в R по різному. Наприклад, як функція, яка обернена до тангенсу:

```
> ctg <- 1/tan(45*pi/180)
> ctg
[1] 1 .
```

або за допомогою функцій синуса та косинуса. Зауважимо, що у цьому прикладі ctg використовується просто як ідентифікатор.

Функція арксинуса позначається в R як **asin()**. Застосовується, якщо необхідно дізнатися синус якого кута (у радіанах) дорівнює заданому числу. Наприклад, синус якого кута дорівнює 0.5? Виконуємо в R:

```
> a <- asin(.5)
> a
[1] 0.5235988
> a_grad <- a*180/pi
> a_grad
[1] 30 .
```

У даних командах знаходиться кут a = 0.5235988, синус якого дорівнює 0.5 у радіанах. Тобто $\sin(0.5235988) = 0.5$. Далі цей кут переводиться у градусну міру: a_grad=30.

Функція арккосинуса позначається в R як **acos()**. Наприклад, знайдемо кут, косинус якого дорівнює -0.5:

```
a <- acos(-0.5)
> a
[1] 2.094395
> a_grad <- a*180/pi
> a_grad
[1] 120 .
```

Шуканий кут дорівнює 2.094395 радіан або 120 градусів.

Функція арктангенса позначається в R як atan(). Наприклад, знайдемо кут, тангенс якого дорівнює одиниці:

```
> a <- atan(1)
> a
[1] 0.7853982
> a_grad <- a*180/pi
> a_grad
[1] 45 .
```

Тобто цей кут дорівнює 0.7853982 радіан або 45 градусів.

Функції арккотангенса в R не існує (як і *котангенса*). Позначимо цю функцію arcctg(x). Тоді значення арккотангенса в R можна обчислити за допомогою формули:

$$arcctg(x) = \begin{cases} -\pi/2 - arctg(x), x < 0 \\ \pi/2 - arctg(x), x \geq 0 \end{cases}.$$

Наприклад, знайдемо значення арккотангенса при x = 1 (acot):

```
acot <- pi/2-atan(1)
> acot
[1] 0.7853982
> acot_grad <- acot*180/pi
> acot_grad
[1] 45 .
```

За такої умови використовується другий рядок формули (для $x \geq 0$). Аналогічно, знайдемо значення арккотангенса (acot) при x = –1:

```
> acot <- -pi/2-atan(-1)
> acot
[1] -0.7853982
> acot_grad <- acot*180/pi
> acot_grad
[1] -45 .
```

За такої умови використовується другий рядок формули (для $x \geq 0$). Функція арккотангенс може бути обчислена також за формулою:

$$arcctg(x) = arctg(1/x),\ x \neq 0.$$

Наприклад, *arcctg*(–0.5) = *arctg*(–2):

```
> a <- atan(-2)
> a
[1] -1.107149
> a_grad <- a*180/pi
> a_grad
[1] -63.43495 .
```

Тут, значення arcctg(–0.5): a <- atan(–2) = –1.107149 у радіанах або –63.43495 у градусах. Перевіримо отриманий результат. Котангенс кута –1.107149 (у радіанах) повинен дорівнювати –0.5. Перевіряємо в R:

```
ctg <- cos(-1.107149)/sin(-1.107149)
> ctg
[1] -0.4999996 .
```

У даному прикладі котангенс обчислюється як синус, поділений на косинус. Результат –0.4999996 є машинним аналогом значення –0.5.

1.4. Вектори в R

1.4.1. Створення векторів

Нульовий вектор заданої довжини (розміру) створюється за допомогою функції **vector()**. Водночас слід задати розмір вектора (число) та його тип:

vector(mode = "*тип даних*", *довжина*) .

У якості *типу даних* можна використовувати: числовий – numeric; логічний – logical; комплексний – complex; символьний – character. Наприклад,

```
> v <- vector("numeric",5)
> v
[1] 0 0 0 0 0
> v <- vector("logical",5)
> v
[1] FALSE FALSE FALSE FALSE FALSE
> v <- vector("complex",5)
> v
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
> v <- vector("character",5)
> v
[1] "" "" "" "" "" .
```

Альтернативою функції **vector()** є функції **numeric()**, **logical()**, **complex()** та **character()**, які створюють вектори окремого типу:

```
> numeric(5)
[1] 0 0 0 0 0
> logical(5)
[1] FALSE FALSE FALSE FALSE FALSE
> complex(5)
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
> character(5)
[1] "" "" "" "" "" .
```

Після створення цих базових векторів, їх заповнюють конкретними значеннями, застосовуючи цикли, або вручну.

Іншим способом створення векторів є використання функції конкатенації **c()**. Водночас компоненти вектора записуються вручну як аргументи цієї функції. Наприклад:

Для числових даних:

```
> x <- c(1,-1,2,-2,3);x
[1]  1 -1  2 -2  3 .
```

Для логічних даних:

```
> x <- c(T,F,F,T,T);x
[1]  TRUE FALSE FALSE  TRUE  TRUE .
```

Для символьних даних:

```
> x <- c("a","d","s","r","w");x
[1] "a" "d" "s" "r" "w".
```

Якщо компоненти вектора мають різні типи, то їх приводять до єдиного типу:

Числовий та логічний типи приводять до числового типу:

```
> x <- c(1,3,F,T,-1);x
[1]  1  3  0  1 -1 # результат.
```

Числовий, логічний та символьний типи приводять до символьного типу:

```
> x <- c(1,3,F,T,-1,"a","b");x
[1] "1"     "3"     "FALSE" "TRUE"  "-1"    "a"     "b".
```

Дійсні та комплексні типи приводять до комплексного типу:

```
> x <- c(1,3,1-2i);x
[1] 1+0i 3+0i 1-2i # результат.
```

Аргументами функції **c()** можуть також бути вектори:

```
> x <- c(1,3,1-2i)
> y <- c(4,5,1+2i)
> z <- c(-1,x,y,1);z
[1] -1+0i  1+0i  3+0i  1-2i  4+0i  5+0i  1+2i  1+0i         # результат.
```

Досить зручним способом створення векторів *із числовими компонентами* є запис із клавіатури за допомогою функції **scan()**:

```
> x <- scan()  # Натисніть Enter
1: 1 2.2 3 4.4 # наберіть на клавіатурі, та натисніть Enter
5:             # натисніть Enter ще раз
Read 4 items
> x
[1] 1.0 2.2 3.0 4.4 # результат.
```

Часто для створення вектора зручно використати функцію повторення **rep()**. Наприклад,

```
> q <- rep(0,5);q
[1] 0 0 0 0 0 # результат.
> q <- rep(1,5);q
[1] 1 1 1 1 1 # результат.
> q <- rep(1+i,5);q
Error: object 'i' not found
> q <- rep(1+1i,5);q
[1] 1+1i 1+1i 1+1i 1+1i 1+1i # результат.
```

Тобто програма **rep()** дозволяє повторювати числа будь-якого класу (*числового, логічного, символьного або комплексного*).

Слід звернути увагу на те, що комплексні числа в R задаються у стандартному вигляді: $a+bi$, але мнима частина повинна бути записана саме у послідовності bi та число b не повинно бути відсутнім. Тобто число $a+i$ необхідно записувати в R як $a+1i$, що й показано у прикладі.

Необхідно мати на увазі, що R використовує принцип "Lazy calculation" де це можливо. Для векторів це означає, що R не розрізняє стовпчики або рядки, поки у цьому нема необхідності.

Якщо є необхідність створити саме стовпчик, можна використати функцію **matrix()**. Наприклад,

```
> a <- matrix(0,1,3);a # Рядок із нулів розміру n = 3
     [,1] [,2] [,3]
[1,]    0    0    0
> length(a)
[1] 3

> a <- matrix(0,3,1);a # Стовпчик із нулів розміру n = 3
     [,1]
[1,]    0
[2,]    0
[3,]    0
> length(a)
[1] 3 # результат.
```

У разі використання векторів необхідно знати його довжину, тобто число його компонент. Для цього існує функція **length()**. Прикладом є попередні "скрипти", а також наступний "скрипт" для компонент будь-якого класу:

```
> x <- c(1,3,F,T,-1,"a","b");x
[1] "1"     "3"     "FALSE" "TRUE"  "-1"    "a"     "b"
> s <- length(x)
> s
[1] 7 # результат.
```

З даного прикладу видно, що ця функція застосовна для векторів будь-якого класу. Але слід пам'ятати, що функція **length()** не розрізняє стовпчики та рядки.

Слід також знати, що в R, як і в Matlab'і існує можливість задавати вектор, як еквідистантну систему точок на певному інтервалі. Така можливість здійснюється функцією **seq()**. Наприклад, на відрізку [0,1] необхідно задати систему точок через крок 0.2. Ця система точок на числової осі складає вектор (*x*):

```
> x=seq(0,1,0.2)
> x
[1] 0.0 0.2 0.4 0.6 0.8 1.0 # результат.
```

1.4.2. Звернення до координат векторів

Під час оброблення економічної інформації та її аналізі необхідно як змінювати певні дані, так і вибирати або додавати певну частину даних. Оскільки усі дані в економіці є векторами з назвою, або без назви, то звернення до даних є, по суті, звернення до координат векторів. Тому розглянемо основні методи зміни та вибору координат вектора в R.

Дуже часто необхідно вибрати з вектора вихідних даних тільки частину даних. Для цього в R існує багато можливостей. Наприклад,

```
> x <- 1:10;x # вихідні дані
 [1]  1  2  3  4  5  6  7  8  9 10
> x1 <- x[x<5];x1 # дані менші за 5
[1] 1 2 3 4 # результат.
> x2 <- x[x<=5];x2 # дані менші, або рівні 5
[1] 1 2 3 4 5 # результат.
> x3 <- x[3:7];x3 # дані від 3 до 7
[1] 3 4 5 6 7 # результат.
```

Координата вектора. Конкретна координата вектора x позначається як $x[i]$ (i-та координата). Тобто перша координата вектора x позначається як $x[1]$, друга координата – як $x[2]$, остання координата – як $x[length(x)]$. Наприклад,

```
> x <- 1:5/5;x # вихідні дані
[1] 0.2 0.4 0.6 0.8 1.0
> x[1]
[1] 0.2 # результат.
> x[length(x)]
[1] 1 # результат.
```

1.4.3. Індексація векторів

Нумерація елементів вектора починається з одиниці. Це завжди треба мати на увазі, бо часто в R необхідно нумерувати вектора, які починаються з нульового індексу, наприклад вектор коефіцієнтів регресії: $b=(b_0,b_1,...,b_m)$. У цьому разі в R необхідно здвинути нумерацію.

Як вже було відмічено у пункті 1.4.2., *i*-та координати вектора b визначається як $b[i]$. Так, перша координата вектора b, $b[1]=b_0$; друга координата, $b[2]=b_0$, і т. і.

За допомогою індексів можна також створювати вектор заданої довжини наступним чином:

```
> a <- 0;a[7] <- 0;a # створюється вектор довжини 7
[1]  0 NA NA NA NA NA  0 # далі можна заповнювати координати іншими числа
ми:
> a[2:6] <- 1:5;a
[1] 0 1 2 3 4 5 0
# або:
> a <- 1:7;a
[1] 1 2 3 4 5 6 7
> a[7] <- 9;a
[1] 1 2 3 4 5 6 9 # і т.і.
```

Звернення до від'ємної координати вектора виключає цю координату:

```
> a <- c(1:7,NA);a
[1]  1  2  3  4  5  6  7 NA
# виключаємо останню (восьму) координату:
> b <- a[-8];b
[1] 1 2 3 4 5 6 7 # але це вже буде вектор з іншим
# ідентифікатором.
```

1.4.4. Функція which()

Ця функція призначена для вибору з вектора даних частини даних, тобто частини вектора за певними умовами. До того ж функція **which(**"умова"**)** визначає саме ті індекси, для яких виконується ця "умова". Наприклад,

```
> a <- c(1,4,33,2,12,8,9,10,23);a
[1]  1  4 33  2 12  8  9 10 23
> i <- which(a<=12);i
[1] 1 2 4 5 6 7 8 # номери координат вектора a
> i <- which(a<12);i
[1] 1 2 4 6 7 8 # номери координат вектора a
> i <- which(a>12);i
[1] 3 9 . # номери координат вектора a
> i <- which(a>=2&a<8);i
[1] 2 4 # номери координат вектора a
> a <- a[i] # частка вектора a з тим самим ідентифікатором

# тобто вектор a перетворився на інший вектор a(4,2);
# операція називається переприсвоєння
[1] 4 2
# якщо вектор a необхідно залишити, то треба використати
# інший ідентифікатор:

> a <- c(1,4,33,2,12,8,9,10,23);a
[1]  1  4 33  2 12  8  9 10 23
> i <- which(a>=2&a<8);i
[1] 2 4
> b <- a[i];b
[1] 4 2 # новий вектор b
> a
[1]  1  4 33  2 12  8  9 10 23 # вектор a залишився
```

Під час економетричних досліджень часто потрібно знати інтервал варіації випадкової змінної, який можна знайти за допомогою функцій **max()** та **min()** як $[\min, \max]$. Номери координат вектора даних, які дають мінімум та максимум можна знайти за допомогою функцій which.min() та which.max(). Наприклад,

```
> a
[1]  1  4 33  2 12  8  9 10 23
> min(a)
[1] 1
> max(a)
[1] 33
```

```
# координати мінімальних та максимальних значень:
> which.min(a)
[1] 1
> which.max(a)
[1] 3
# різні варіанти знаходження мінімальних та максимальних значень:
> mina <- a[which.min(a)]
> mina
[1] 1
> maxa <- a[which.max(a)]
> maxa
[1] 33
> a[1]
[1] 1
> a[3]
[1] 33 .
```

1.4.5. Маніпуляції з векторами

До векторів в R можна застосовувати як арифметичні, так і функціональні оператори.

Арифметичні дії над векторами:

```
> a <- c(1,2,3,4) # завдання вектора
> b <- c(5,6,7,8)
> q <- a+b;q      # складання
[1]  6  8 10 12
> q <- a-b;q      # віднімання
[1] -4 -4 -4 -4
> q <- 2*a-3*b;q  # множення на число
[1] -13 -14 -15 -16
> q <- a/3-2*b;q  # ділення на число
[1]  -9.666667 -11.333333 -13.000000 -14.666667 # результат.
```

Функціональні оператори застосовуються до кожної компоненти вектора окремо. Наприклад,

```
> x <- c(0,pi/6,pi/3,pi/2,pi)
> y <- sin(x);y # Обчислення синусу
[1] 0.000000e+00 5.000000e-01 8.660254e-01 1.000000e+00 1.224606e-16
> x <- c(1,2,3,4,5)
> y <- log10(x);y # обчислення десятинного логарифму
[1] 0.0000000 0.3010300 0.4771213 0.6020600 0.6989700
> x <- c(1,2,3,4,5)
> y <- x^2;y    # зведення в ступень
[1]  1  4  9 16 25
> z <- sqrt(y);z # добування кореня
[1] 1 2 3 4 5
> z <- y^(1/2);z # або зведення в ступень
[1] 1 2 3 4 5 # результат.
```

Скалярний добуток векторів. Для обчислення скалярного добутку в R застосовується не дуже зручне позначення, %*%:

```
> x <- c(1,2,3,4,5)
> y <- c(2,3,4,5,6)
> z <- x%*%y;z
     [,1]
[1,]   70 # результат.
```

З іншого боку, для поелементного множення застосовується звичайне позначення, "*":

```
> x <- c(1,2,3,4,5) # кожна компонента вектора x
> y <- c(2,3,4,5,6) # множиться на відповідну компоненту
> z <- x*y;z        # вектора y
[1]  2  6 12 20 30 # результат.
```

Важливе значення в економетричних дослідженнях мають функції, які визначають максимальний та мінімальний елемент вектора даних, тобто функції **max()** та **min()**. Наприклад,

```
> x <- rpois(7,1.5);x #випадкова величина з розподілу Бернулі
[1] 1 2 5 1 1 0 2
> maxx <- max(x);maxx # максимальне значення x (5)
[1] 5
> minx < - min(x);minx # мінімальне значення x (0)
[1] 0 # результат.
```

Також важливо знати середнє значення компонентів вектора, оскільки це є середнє значення економічного показника в економетричному аналізі. Середнє значення в R визначається за допомогою функції **mean()**. Наприклад,

```
> q # вектор даних містить величину NA
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7  NA 3.4
> mean(q) # у цьому разі функція mean() видає неінформативну
# відповідь:
[1] NA
# Необхідно убрати значення NA з даних. Для цього необхідно
# знайти довжину вектора q:
> length(q)
[1] 12
# Дали знаходимо вектор до величини NA:
> q1 <- q[1:10];q1
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7
```

```
# Після цього складаємо новий вектор за допомогою функції
# конкатенації c():
> q2 <- c(q1,3.4)
> q2 # Вектор q2 вже не містить величини NA:
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7 3.4
# Знаходимо середнє значення (meanq2 є ідентифікатором серед
# нього значення):
> meanq2 <- mean(q2);meanq2
[1] 3.281818 # результат.
```

Величина NA у даному прикладі виникає у тому разі, коли значення економічного показника або пропущене, або не має сенсу. Його можна убрати, як це й зроблено, а можна вставити замість нього інше значення після ретельного дослідження вихідних даних. Припустимо, дослідник вирішив замінити значення NA на 2.7. Це можна зробити наступним чином:

```
> q
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7  NA 3.4
> q[11] <- 2.7 # координата q(11) замінена на 2.7
> q
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7 2.7 3.4
> meanq <- mean(q);meanq
[1] 3.233333 # результат.
```

Якщо ми хочемо залишити величину NA у векторі *q* та підрахувати середнє значення без NA, слід використати функцію **mean()** з опцією: **na.rm=TRUE**:

```
> meanq <- mean(q,na.rm=T);meanq
[1] 3.281818 # результат.
```

Велике значення у статистиці та економетриці має поняття розкиду варіації вибірки (sample range). Цю величину в R можна розрахувати за допомогою функції **range()**. Наприклад,

```
> q # вихідні дані
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7  NA 3.4
> range(q)
[1] NA NA
> range(q,na.rm=F) # те саме, що й range(q)
[1] NA NA
> range(q,na.rm=T) # розраховує розкид варіації не звертаючи
уваги на наявність величини NA
[1] 2.4 4.1 # дані знаходяться у інтервалі [2.4;4.1] .
```

Часто необхідно підрахувати просто суму елементів вектора даних. Для цього існує функція **sum()**. Наприклад,

```
> q <- c(2.5 ,3.6, 3.8, 4.1, 3.7, 2.8, 3.2, 2.4, 2.9, 3.7,   NA, 3.4)
# вихідні дані містять NA
> q
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7  NA 3.4
> sq <- sum(q,na.rm=T);sq # застосовується опція: na.rm=T
[1] 36.1
> q1[11] <- 2.7 # замінюємо NA на 2.7
> q1
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7 2.7 3.4
> sq1 <- sum(q1);sq1 # опція na.rm=T не застосовується
[1] 38.8
# сума квадратів елементів вектора (SS):
> q1 <- q1^2;q1 # елементи вектора q1 зводяться в квадрат
 [1]  6.25 12.96 14.44 16.81 13.69  7.84 10.24  5.76  8.41   13.69  7.29 11.56
> SS <- sum(q1);SS # сума квадратів
[1] 128.94 # результат.
```

У даному прикладі показано також як розраховувати дуже важливу в економетриці величину SS – суму квадратів елементів вектора даних.

Аналогічно розрахунку суми компонент вектора можна розрахувати також добуток його компонент, за допомогою функції **prod()**. Наприклад,

```
> q # вихідні дані
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7  NA 3.4
> prod(q) # команда спрацює, якщо нема величин NA та NaN
[1] NA # відповідь, коли є величини NA та NaN
> prod(q,na.rm=T) # опція na.rm=T дає можливість розрахувати
# добуток координат вектора без урахування величин NA та NaN
[1] 407014.2 # результат.
```

Існує можливість сортування даних, які записані у векторному вигляді за допомогою команди **sort()**.

```
> y # вихідні дані
[1]  -4.957011 -13.178933  -7.803061 -14.127616 -13.495161 -17.289894 -24.246905
> sort(y) # сортування за зростанням
[1] -24.246905 -17.289894 -14.127616 -13.495161 -13.178933  -7.803061  -4.957011
> sort(y,decreasing=TRUE) # сортування за зменшенням
[1]    -4.957011    -7.803061 -13.178933 -13.495161 -14.127616 -17.289894
-24.246905 # результат.
```

Для сортування компонент вектора даних за зменшенням можна використати також команду **rev(sort())**. Наприклад,

```
> r <- c(4.9570, 13.1789,  7.8030, 14.1276, 13.4951, 17.2898, 24.2469)
# вихідні дані
[1]  4.9570 13.1789  7.8030 14.1276 13.4951 17.2898 24.2469
> rs <- rev(sort(r));rs # сортовані дані
[1] 24.2469 17.2898 14.1276 13.4951 13.1789  7.8030  4.9570
# результат.
```

Зауважимо, що ця команда ще може зустрітися у літературі, але, як правило, використовується команда **sort(r,decreasing=TRUE)**.

У економіці дуже часто використовується процес ранжування даних за зростанням. Якщо усі дані різні, то можна використати просто функцію **rank()**. Наприклад,

```
> a <- c(5,8,3,9);a
[1] 5 8 3 9
> r <- rank(a)
> r
[1] 2 3 1 4 .
```

Однак у економіці часто необхідно ранжувати об'єкти оцінки яких не усі різні. У такому випадку слід використовувати функцію **rank()** з опціями, які визначають метод розрахунку рангів:

```
> a <- c(5,8,3,9,9,6,5,1);a
[1] 5 8 3 9 9 6 5 1
> r <- rank(a,ties='average');r# метод середніх для однакових
# елементів
[1] 3.5 6.0 2.0 7.5 7.5 5.0 3.5 1.0
> r <- rank(a,ties='first');r # метод першого елемента для однакових елементів
[1] 3 6 2 7 8 5 4 1
> r <- rank(a,ties='max');r # метод максимального елемента
# для однакових елементів
[1] 4 6 2 8 8 5 4 1
> r <- rank(a,ties='min');r # метод мінімального елемента для
# однакових елементів
[1] 3 6 2 7 7 5 3 1 .
```

Додатковий аргумент (**na.last**) відповідає за наявність значень NA. Наприклад,

```
> a <- c(5,8,3,9,9,6,NA,5,1);a
[1]  5  8  3  9  9  6 NA  5  1
> r <- rank(a,ties='min',na.last=T);r
[1] 3 6 2 7 7 5 9 3 1 .
```

Функція **match()** застосовується, коли необхідно дізнатися, на якому місці у векторі *b* з'являються елементи вектора *a*. Наприклад,

```
> a <- c(5,4,4,3);a
[1] 5 4 4 3
> b <- c(5,8,3,9);b
[1] 5 8 3 9
> match(a,b)
[1]  1 NA NA  3 # перша координата вектора a (a1) співпадає перший раз із координатою вектора b на першому місці (1). Друга координата вектора a (a2) не співпадає з жодною координатою вектора b (NA). Також третя координата вектора a (a3) не співпадає з жодною координатою вектора b (NA). Четверта координата  вектора a (a4) співпадає з третьої координатою вектора b (3)
> match(b,a)
[1]  1 NA  4 NA # аналогічно для координат вектора b.
```

Кумулятивні функції вектора. До цих функцій відносяться наступні функції: **cumsum()**, **cumprod**(), **cummax()** та **cummin()**. Функції **cumsum()** та **cumprod**() являють собою кумулятивну суму та кумулятивний добуток. Наприклад,

```
> q <- 1:5;q
[1] 1 2 3 4 5
> q1 <- cumsum(q);q1
[1]  1  3  6 10 15
> q2 <- cumprod(q);q2
[1]   1   2   6  24 120 .
```

Функції cummax() та cummin():

```
> q <- -3:3;q
[1] -3 -2 -1  0  1  2  3
> q1=cumprod(q);q1
[1] -3  6 -6  0  0  0  0
> q2 <- cummin(q1);q2 # послідовний мінімум вектора q2:
# min(-3)=-3; min(-3,6)=-3; min(-3,6,-6)=-6; min(-3,6,-6,0)= # -6;
min(-3,6,-6,0,0)=-6; …
[1] -3 -3 -6 -6 -6 -6 -6
> q3 <- cummax(q1);q3 # послідовний максимум вектора q2:
# max(-3)=-3; max(-3,6)=6; max(-3,6,-6)=6; max(-3,6,-6,0)=-6; …
[1] -3  6  6  6  6  6  6
```

Велике значення під час оброблення економічних даних мають логічні вектори, елементи яких є TRUE (T), FALSE (F) або NA. Логічні вектори виникають під час операцій порівняння та застосування логічних функцій.

Наприклад, ми маємо дані про вік групи людей:

```
> age <- c(12, 12, 32, 23, 43, 15, 18, 73, 38);q
[1] 12 12 32 23 43 15 18 73 38
```

Припустимо, що за якісною шкалою до категорії молодого віку відносяться люди за віком більше, ніж 10 років та менше, ніж 30 років. Перевіряємо, які респонденти (по порядку) задовольняють цим умовам:

```
> young <- age>15&age<30;young
[1] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
```

Логічний вектор young показує, що умовам відповідають: четвертий та сьомий респонденти.

Для того, щоб перетворити логічний вектор на числовий достатньо помножити його на одиницю:

```
> y <- 1*young;y
[1] 0 0 0 1 0 0 1 0 0
```

Часто необхідно знайти саме які респонденти за номерами задовольняють поставленої умові. Тобто знайти номера відповідних координат вектора age. Це можна зробити за допомогою функції **which()** наступним чином:

```
> y <- 1*young;y
[1] 0 0 0 1 0 0 1 0 0
> which(y!=0) # ця функція знаходить номера ненульових координат
[1] 4 7
> which(y==0) # ця функція знаходить номера нульових координат
[1] 1 2 3 5 6 8 9
> y[4] # ненульові координати вектора y
[1] 1
> y[7] # ненульові координати вектора y
[1] 1
> age[4] # ненульова координата вектора age є 4, її значення
# 23[1] 23
> age[7] # ненульова координата вектора age є 7, її значення
# 18[1] 18
```

Оскільки вектори даних не завжди доступні перегляду, важливо з'ясувати, чи є серед них значення типу NA або NaN. Для цього існують функції **is.na(**) та **is.nan()**. Хоча вони різні за іменем, вони обидві виконують

одно й теж: перевіряють, є чи нема серед координат вектора значення типу NA NaN. Наприклад,

```
> q <- c(1,-2,2,Inf,NA,12,NaN);q # вектор даних містить Inf, NA та NaN
[1]   1  -2   2 Inf  NA  12 NaN
> is.na(q)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
> is.nan(q)
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
> is.infinite(q)
[1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

Як можна побачити із цього прикладу, **is.na()** та **is.nan()** не можуть розпізнати наявність величин типу Inf (тобто особливості). Для цього існує функція **is.infinite()**, що показано на тому самому прикладі.

Слід зауважити, що є також функція **is.infinite()**, яка ідентифікує як величин типу Inf, так і величини типу NA або NaN. Наприклад,

```
[1]   1   3   5 Inf  NA   6   7
> is.finite(q)
[1]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE .
```

Ці функції не є зручними у випадку великих вибірок тому що результативний вектор має той самий розмір, що й вихідний вектор. У цьому разі слід використати функцію **anyNA()**. Наприклад,

```
> q <- c(1,3,5,Inf,NA,6,7);q
[1]   1   3   5 Inf  NA   6   7
> anyNA(q)
[1] TRUE # є наявність величин типу NA або NaN.
```

Але ця функція не ідентифікує величини типу Inf:

```
[1]   1   3   5 Inf   6   7
> anyNA(q)
[1] FALSE
```

Якщо ми маємо великий вектор даних та хочемо перевірити його на наявність координат типу Inf, NA або NaN, можна це зробити наступним чином:

```
> q <- c(1,3,5,Inf,NA,6,7,Inf,NaN);q
[1]   1   3   5 Inf  NA   6   7 Inf NaN
> q[!is.finite(q)]
[1] Inf  NA Inf NaN .
```

У цьому разі у відповіді присутні стільки позначок, скільки координат такого типу є у вектора даних.

Часто необхідно замінити Inf на NA, бо, як правило, функції R не обробляють дані з величинами типу Inf. Це можна зробити наступним чином:

```
> q <- c(1,3,5,Inf,6,7);q
[1]   1   3   5 Inf   6   7
> q[!is.finite(q)] <- NA
> q
[1]  1  3  5 NA  6  7 .
```

Після заміни Inf на NA можливо виконувати функції R над вектором. Наприклад, обчислити середню:

```
> q <- c(1,3,5,6,7,NA);q
[1]  1  3  5  6  7 NA
> q <- t(t(q)) # створення стовпчику
> colMeans(q,na.rm=T) # взагалі, це матрична функцію
[1] 4.4 # усереднення по п'яти координатах замість шести.
```

Якщо під час перевірки даних цими функціями отримана хоча б одна відповідь "TRUE", необхідно або убирати ці дані, або як-то їх нейтралізувати. Якщо дослідник вважає, що деякі дані необхідно просто убрати, то для цього необхідно знайти номера індексів цих координат. Однак при цьому необхідно пам'ятати, що елімінація даних невигідна, тому що зменшується розмір вибірки. Саме тому в R існують функції з опціями типу na.rm=T, які дозволяють виконувати статистичні розрахунки з пропущеними даними або з аутлайєрами, які інтерпретуються як величини типу NA.

Як правило, дані в R мають назву, що більш детально буде розглянуто далі. Зараз звернемо увагу тільки на один аспект. Часто елемента вектора є результати попередніх розрахунків. Наприклад є вектор середніх значень за групами об'єктів. Іноді має сенс прив'язати до цього вектора назви об'єктів. Наприклад, маємо дані середньої ваги студентів за дев'ятьма групами (вектор **q**). Додаємо до них назви груп за допомогою функції **names()**:

```
> q <- c(63,64,60,71,65,66,70,68,72);q
[1] 63 64 60 71 65 66 70 68 72
> names(q) <- c('G1','G2','G3','G4','G5','G6','G7','G8','G9')
> q
G1 G2 G3 G4 G5 G6 G7 G8 G9
63 64 60 71 65 66 70 68 72 .
```

Звертаємось до координат вектора q:

```
> q[5]
G5
65
> q[G5]# так не можна звертатися до вектора!
Error: object 'G5' not found
> q['G5']
G5
65 .
```

З іменованими векторами можна виконувати усі математичні операції, але при цьому назви можуть змінюватися:

```
> q[3]*q[4]
  G3
4260
> q[3]+q[4]
 G3
131
> log(q[5])
      G5
4.174387
> e <- cumsum(q);e
 G1  G2  G3  G4  G5  G6  G7  G8  G9
 63 127 187 258 323 389 459 527 599
> e/q
   G1      G2      G3      G4      G5      G6      G7      G8      G9
1.000   1.984   3.116   3.633   4.969   5.893   6.557   7.750   8.319
```

Для того, щоб зняти назви достатньо застосувати функцію **names()** ще раз із пустими назвами:

```
1 2 3
> q <- c(1,2,3);q
[1] 1 2 3
> names(q) <- c('a','b','c');q
a b c
1 2 3
> names(q) <- c('','','');q
1 2 3 .
```

1.5. Матриці в R

1.5.1. Завдання числової матриці. Функція matrix()

Числову матрицю можливо задати із числового вектора за допомогою функції **matrix()**. Для цього необхідно розрахувати число рядків

та стовпчиків. При цьому добуток числа рядків на число стовпчиків повинно дорівнювати довжині вектора. Стандартне означення цієї функції:

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL).
```

Для завдання матриці необхідно замість "Data" записати назву масиву даних, необхідно вказати число рядків nrow = m та /або число стовпчиків ncol = n (за замовчанням, число рядків та число стовпчиків дорівнює одиниці). Далі слід визначити як елементи вектора "Data" заповнюють матрицю – або по рядкам, або по стовпчикам (за замовчанням, матриця заповнюється по стовпчикам). До того ж не обов'язково задавати одночасно число рядків та стовпчиків. Аргумент dimnames – список із двох компонент, перша з яких задає назву рядків, а друга – назву стовпчиків (за замовчанням назви не задаються). Наприклад,

```
> A <- matrix(1:6,2)# З вектора 1:6 складається матриця
# з двох рядків послідовно по стовпчикам за замовчанням.
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> A <- matrix(1:6,2,byrow=T);A # З вектора 1:6 складається
# матриця з двох рядків послідовно по рядкам.
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

> A <- matrix(1:6,3,byrow=T);A# З вектора 1:6 складається
# матриця з трьох рядків послідовно по рядкам.
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6.
```

Озаглавити стовпчики можна відразу, коли задається матриця, наприклад,

```
> A <- matrix(1:6,3,byrow=T,dimnames=list(c(1,2,3),c('Aa','Bb')));A
  Aa Bb
1  1  2
2  3  4
3  5  6,
```

а можна це зробити після завдання матриці:

```
> A <- matrix(1:6,3,byrow=T);A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> dimnames(A) <- list(c(1,2,3),c('Aa','Bb'))
> A
  Aa Bb
1  1  2
2  3  4
3  5  6.
```

Слід зауважити, що під час озаглавлювання стовпчиків числова матриця перетворюється на символьну.

Матрицю можна заповнювати множенням векторів. Наприклад,

```
> a <- c(1,2,3);a # Перший вектор
[1] 1 2 3
> t(a) # Перший вектор як матриця
     [,1] [,2] [,3]
[1,]    1    2    3
> t(t(a)) # Перший вектор як транспонована матриця
     [,1]
[1,]    1
[2,]    2
[3,]    3
> b <- c(4,5,6);b # Другий вектор
[1] 4 5 6

> t(t(a))%*%b # Добуток стовпчика на рядок дає матрицю:
     [,1] [,2]  [,3]
[1,]    4    5    6
[2,]    8   10  12
[3,]  12   15  18
> t(t(a))
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

Часто, перш ніж заповнювати матрицю числами, її заповнюють символами NA або нулями. Наприклад,

```
M=matrix(,nrow=4,ncol=3)
> M
```

```
     [,1]  [,2]  [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA
```

matrix(,nrow=4,ncol=5) – створює матрицю розміру 4 x 5 заповнену NA (preallocation). Далі заповнюють стовпчики, наприклад нулями:

```
> M[,1]=vector(length=4,mode='numeric') # заповнення нулями
> M
     [,1]  [,2]  [,3]
[1,]    0   NA   NA
[2,]    0   NA   NA
[3,]    0   NA   NA
[4,]    0   NA   NA
```

Далі заповнюється другий стовпчик: M[,2] і т. ін.

Матрицю можна заповнювати будь-якими числами покомпонентно, наприклад:

```
> a <- matrix(,nrow=3,ncol=3)
> a
     [,1]  [,2]  [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
> a[1,1] <- 2 # заповнюється компонента (1,1)
> a
     [,1]  [,2]  [,3]
[1,]    2   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
> a[1,] <- c(2,3,4) # заповнюється перший рядок

> a
     [,1]  [,2]  [,3]
[1,]    2     3    4
[2,]   NA   NA   NA
[3,]   NA   NA   NA.
```

Матрицю можна заповнювати числами або символами NA. Якщо матриця вже заповнена, але має числа типу Inf, то ці числа необхідно замінити на NA для того, щоб можна було виповнювати статистичні операції незважаючи на наявність невизначених чисел. Це можна зробити наступним чином:

```
> M=matrix(,nrow=4,ncol=3);M
     [,1]  [,2]  [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA
> M[1,2] <- Inf  # заміняємо в (1,2) NA на Inf
> M
     [,1]  [,2]  [,3]
[1,]   NA   Inf   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA
> M[!is.finite(M)] <- NA  # заміняються всі Inf на NA
> M
     [,1]  [,2]  [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA.
```

Ці операції дозволяють обчислювати статистичні характеристики стовпчиків або рядків спостережених даних, якщо вони мають нечислові значення, наприклад, обчислення середньої без урахування NA:

```
> a
     [,1]  [,2]  [,3]
[1,]    2    3     4
[2,]   21   31   NA
[3,]   21   31   12
> colMeans(a,na.rm=T)
[1] 14.66667 21.66667  8.00000 # середні значення за стовпчиками;
у третьому стовпчику значення NA не враховується при обчисленні середньої.
```

Для чисельних досліджень використовується завдання елементів матриці у циклі. Наприклад, для матриці Гілберта:

```
> M=matrix(,nrow=3,ncol=3) # preallocation; задається матриця розміру [3,3]
з елементами NA
> for (i in 1:3){for(j in 1:3) {M[i,j]=1/(i+j)}}# елементи NA змінюються на числа
>  M # виклик матриці у вікно виконання (Consoul) або для подальшого
використання:
          [,1]      [,2]      [,3]
[1,] 0.5000000 0.3333333 0.2500000
[2,] 0.3333333 0.2500000 0.2000000
[3,] 0.2500000 0.2000000 0.1666667
# приклад подальшого використання:
> det(M)# детермінант
[1] 2.314815e-05
> kappa(M) # число обумовленості матриці
[1] 1555.098
> rcond(M) # обернене число обумовленості матриці
[1] 0.0004962779
```

Для роботи з матрицями даних необхідно знати число рядків (розмір вибірки), число стовпчиків (число факторів) та взагалі розмірність матриці, що обчислюється функціями: **nrow(A)**, **ncol(A)** та **dim(A),** відповідно. Наприклад,

```
> nrow(M)
[1] 3
> ncol(M)
[1] 3
> dim(M)
[1] 3 3.
```

Дуже важливою є функція **cbind()**, яка об'єднує матриці з однаковим числом рядків за рядками. Ця функція використовується під час модифікації спостережених даних, якщо потрібно додати нові фактори до системи показників. Наприклад,

```
> A <- matrix(1:12,nrow=4);A
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> B <- t(t(3:6));B
     [,1]
[1,]    3
[2,]    4
[3,]    5
[4,]    6

> C <- cbind(A,B);C
     [,1] [,2] [,3] [,4]
[1,]    1    5    9    3
[2,]    2    6   10    4
[3,]    3    7   11    5
[4,]    4    8   12    6.
```

Так само можливо додати будь-яке число спостережених даних до вибірки за допомогою функції **rbind()**. Наприклад, *A* є матриця спостережених даних і розмір вибірки, тобто число рядків матриці *A*, недостатньо для досягнення необхідної значимості статистичних результатів. Тоді, з цієї ж генеральної сукупності вибираються додаткові дані, матриця *B*. Ці дані повинні бути доданими до попередніх даних, тобто до матриці *A*. Результатом буде нова матриця даних, *C*:

```
> A <- matrix(1:12,nrow=3);A
     [,1] [,2] [,3]  [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> B <- matrix(13:24,nrow=3);B
      [,1] [,2]  [,3]  [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24

> C=rbind(A,B);C
     [,1] [,2] [,3]  [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
[4,]  13   16  19   22
[5,]  14   17  20   23
[6,]  15   18  21   24.
```

У статистичних розрахунках, які пов'язані з рішенням задачі регресії, наприклад, необхідно використати діагональну матрицю, яку можна задати за допомогою функції **diag()**:

```
> E <- diag(x=1,3)
> E
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1.
```

За допомогою функції **diag()** зручно об'являти матрицю будь-якого розміру до обчислювання її елементів у циклі:

```
> E <- diag(x=1,3,4)
> E
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0

> E <- diag(x=1,4,3);E
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
[4,]    0    0    0.
```

За допомогою функції **diag()** зручно також створювати нульову матрицю будь-якого розміру:

```
> E <- diag(x=0,4,3);E
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
[4,]    0    0    0
```

Дуже важливим застосуванням цієї функції є виділення елементів головної діагоналі матриці у вигляді вектора:

```
> A <- matrix(1:9,nrow=3);A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> D=diag(A);D
[1] 1 5 9,
```

а також вставляти замість головної діагоналі задані числа:

```
> A <- matrix(1:9,nrow=3);A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> diag(A) <- c(0,0,0)
> A
     [,1] [,2] [,3]
[1,]    0    4    7
[2,]    2    0    8
[3,]    3    6    0.
```

Можна також заповнювати матрицю добутком двох векторів за допомогою функції outer(x,y,'дія'). Наприклад,

```
> x <- 1:3
> y <- 2:6
> z <- outer(x,y,'*');z
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    3    4    5    6
[2,]    4    6    8   10   12
[3,]    6    9   12   15   18
```

де 'дія'=*', тобто є множенням у даному випадку.

1.5.2. Редактор даних

Для редагування даних, які являють собою матриці чисел, в R існує спеціальний редактор. Якщо дані вже існують у вигляді матриці з назвою, наприклад *A*, то редактор викликається за допомогою функції **fix(*A*)**. Якщо дані після редагування під тим самим ім'ям, то викликати редактор слід наступним чином: A = fix(A). Після редагування редактор просто слід закрити. Водночас результат редагування автоматично зберігається з ім'ям *A*. Ясно, що для редагування даних з ім'ям *A* змінна *A* повинна знаходитися у глобальному оточенні (вікно Global Environment). Наприклад, є матриця даних *A*:

```
> A <- matrix(1:9,nrow=3);A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9.
```

Слід замінити в неї третю колонку:

```
> A <- fix(A)
> A # дані після редагування
     col1 col2 col3
[1,]    1    4    3
[2,]    2    5    2
[3,]    3    6    1.
```

Під час виконання команди A <- fix(A) визивається редактор (рис. 1.5.1).

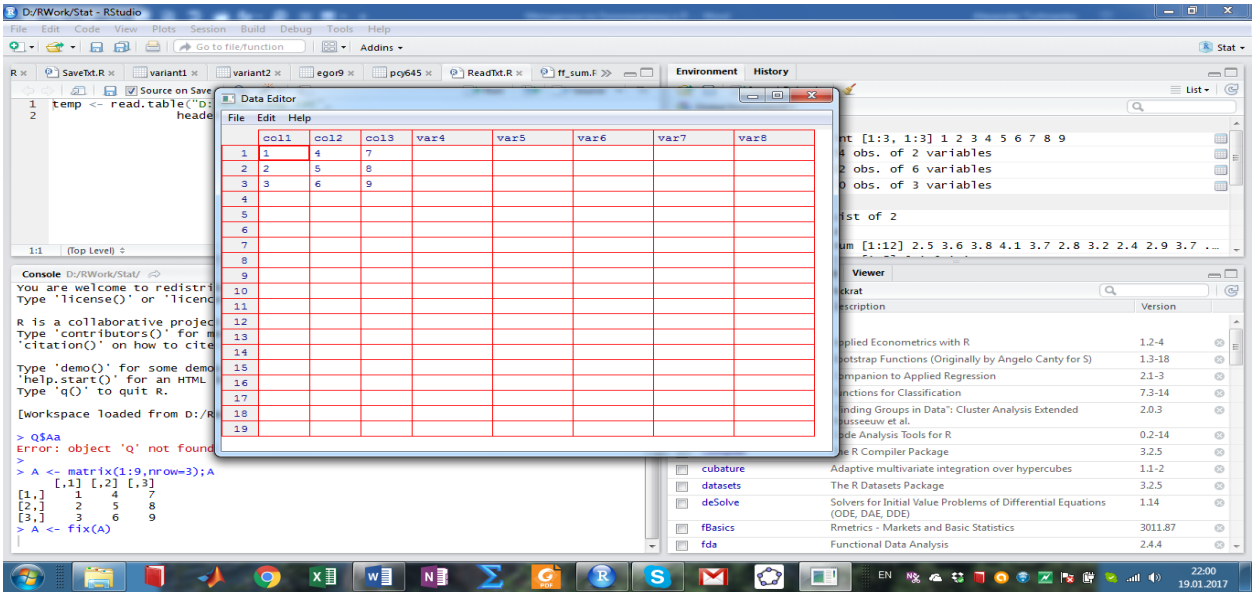


Рис. 1.5.1. **Редактор даних**

У ньому можна змінювати елементи матриці даних як в Excel. Після завершення редагування слід натиснути кнопки **File** та **Close**, або просто закрити вікно редактора червоною кнопкою.

У редакторі можна не тільки редагувати дані, але й безпосередньо набирати дані. Для цього необхідно спочатку задати нульову, наприклад, матрицю, визвати редактор, та змінять дані як у Excel. Наприклад,

```
> A <- diag(x=0,3,4);A # Задається матриця будь-якого розміру
# Далі розмір змінюється безпосередньо у редакторі
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
> A <- fix(A)
> A
     col1 col2 col3 col4 var5
[1,]    2    1    4    2    2
[2,]   -1    3   -4    1    3
[3,]    1   -2    4    6    1
[4,]   -1    4    5    1    6.
```

Зазначимо, що після редактора колонки отримують назви, які можна змінити, якщо клікнути на них лівою кнопкою. Крім того, так же точно встановлюється тип даних: числові або текстові.

1.5.3. Арифметичні дії з матрицями

Арифметичні операції додавання з матрицями здійснюються поелементно, тому розмір матриць повинен бути однаковим. Тобто складати або віднімати можна тільки матриці одного розміру. Ці дії не відрізняються від дій з векторами.

Слід мати на увазі, що до будь-якої матриці можна додати будь-яке число. Це сприймається в R як додавання матриці того ж розміру, усі елементи якої дорівнюють цьому числу. Наприклад,

```
> A <- matrix(1:9,nrow=3);A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> A+3
     [,1] [,2]  [,3]
[1,]    4    7   10
[2,]    5    8   11
[3,]    6    9   12.
```

Будь-яку матрицю можна помножити або розділити на будь-яке при пустиме число. У цьому разі усі елементи матриці помножаться або розді ляться на це число. Наприклад,

```
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> B <- A/3;B
         [,1]          [,2]         [,3]
[1,] 0.3333333 1.333333 2.333333
[2,] 0.6666667 1.666667 2.666667
[3,] 1.0000000 2.000000 3.000000.
```

Множення матриць відповідає звичайним правилам матричної алгебри, але позначаються в R досить незвичайним чином: "**%*%**", на що необхідно звернути увагу, оскільки звичайна позначка "*****" застосовується для поелементного множення, коли добуток двох матриць одного й того ж розміру створюється як матриця, елементи якої є добутком однакових елементів кожної з матриць.

Наприклад,

```
> A <- matrix(1:9,nrow=3);A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> B <- diag(x=1,3,3);B
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1.
```

Звичайний добуток матриць *A* та *B*:

```
> A%*%B
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9.
```

У той самий час поелементний добуток матриць *A* та *B* дає зовсім інший результат:

```
> A*B
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    5    0
[3,]    0    0    9.
```

1.5.4. Функції від матриць

Елементарні функції від матриць виконуються поелементно. Якщо, наприклад матриця має вигляд:

```
   > A <- matrix(1:9,nrow=3);A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9,
```

то можна витягти корінь квадратний з кожного елемента цієї матриці:

```
> B <- sqrt(A);B
         [,1]         [,2]         [,3]
[1,] 1.000000 2.000000 2.645751
[2,] 1.414214 2.236068 2.828427
[3,] 1.732051 2.449490 3.000000.
```

Можливо також усі елементи матриці A піднести до будь-якого ступеня:

```
> B <- A^3;B
     [,1]   [,2]    [,3]
[1,]    1    64   343
[2,]    8  125   512
[3,]   27  216  729.
```

Детермінант матриці визначається за допомогою функції **det()**:

```
> A <- matrix(1:9,nrow=3);A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> det(A)
[1] 0.
```

Для рішення визначеної системи лінійних рівнянь, як наприклад, у задачі регресії, можна використати функцію **solve()**. Функція **solve(A)** визначає обернену матрицю до матриці А. Наприклад,

```
> A # визначник цієї матриці дорівнює нулю; обернена матриця не може бути знайдена
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> diag(A)=c(1,1,1) # змінюємо діагональні елементи матриці А

> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    1    8
[3,]    3    6    1
> det(A)
[1] 104  # визначник не дорівнює нулю

> iA=solve(A) # знаходимо обернену матрицю
> iA
           [,1]          [,2]          [,3]
[1,] -0.45192308  0.36538462  0.24038462
[2,]  0.21153846 -0.19230769  0.05769231
[3,]  0.08653846  0.05769231 -0.06730769

> iA%*%A # перевіряємо правильність обчислень
             [,1]          [,2]          [,3]
[1,]  1.000000e+00 2.775558e-16  4.163336e-16
[2,] -3.469447e-17 1.000000e+00 -2.081668e-17
[3,]  1.387779e-17 2.775558e-17  1.000000e+00.
```

Слід звернути увагу на результат перевірки правильності знаходження оберненої матриці. У результаті виповнення команди iA%*%A ми повинні отримати одиничну матрицю. Те, що отримано, й є одиничною матрицею з машинними нулями замість точних нулів.

1.5.5. Операції з індексами

Звернення до елементів матриці А відбувається як $A[i, j]$. Наприклад, якщо дана матриця А:

```
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    1    8
[3,]    3    6    1,
```

то її елемент $a(1,2)$:

```
> A[1,2]
[1] 4.
```

Якщо індекси від'ємні, то це означає вилучення рядка або стовпчика матриці. Наприклад:

```
  > A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    1    8
[3,]    3    6    1,
```

тоді,

```
> A[-1,-1] # вилучається перший рядок та перший стовпчик матриці A
     [,1] [,2]
[1,]    1    8
[2,]    6    1;

> A[-1,] # вилучається тільки перший рядок матриці A
     [,1] [,2] [,3]
[1,]    2    1    8
[2,]    3    6    1;

> A[,-1] # вилучається тільки перший стовпчик матриці A
     [,1] [,2]
[1,]    4    7
[2,]    1    8
[3,]    6    1;

> A[,-2] # вилучається тільки другий стовпчик матриці A
     [,1] [,2]
[1,]    1    7
[2,]    2    8
[3,]    3    1;

> A[-1,2] # вилучається рядок, а залишається тільки другий
# стовпчик матриці A
[1] 1 6.
```

Індекси матриці також можуть бути векторами. Додатні координати цих векторів дозволяють звернутися до певних елементів матриці, а від'ємні координати дозволяють вилучити відповідні елементи з матриці.

Розділ 2. Робота з даними в R

Метою статистичних та економетричних досліджень є вилучення інформації про характеристики генеральної сукупності економічних об'єктів за спостереженими даними. Найважливішим етапом цих досліджень є початкове оброблення економічних даних за допомогою машинних методів з використанням відповідного програмного забезпечення, між яким R є найбільш популярним на сьогодні.

Розглянемо основні аспекти оброблення спостережених даних в R. Перш ніж обробляти спостережені дані їх необхідно ввести в певному вигляді в Оточення R. Для цього дані повинні бути записаними у спеціальні форми.

2.1. Структури даних в R

Існують три *типи* даних: числові (numeric), текстові (characters) та логічні (logical). Дані в R можна записати в наступних *формах*, а саме **vector;matrix; array; list; data.frame**:

vector – одновимірний масив даних одного й того самого типу;

matrix – двовимірний масив даних одного й того самого типу;

array – матриця даних розміру більше двох;

list – одновимірний масив даних різних типів;

data.frame – двовимірний масив даних, кожна колонка яких складається з даних одного типу, але різні колонки можуть бути даними різних типів; кожна колонка даних має свою назву.

У якості даних можуть бути тільки цифри та букви у лапках, а також слова, які з'єднані підкресленням або точкою: "Факультет_ЕП", або "Факультет.ЕП ". Назва списку не повинна починатися із цифри.

Оскільки запис даних у вектор та матрицю вже розглядали у даній роботі, розглянемо список (**array**) та **data.frame**.

2.1.1. Список

Списки в R є колекціями об'єктів, доступ до котрих можна здійснювати за номером або за назвою. Об'єктами списку можуть бути дані різного типу, а також вектори або списки. Кожен список повинен мати окрему назву латиною. У назві можуть бути також і цифри, але не на першому місці.

Створимо для прикладу список студентів, у якому вказано, факультет, курс, група, бюджет (T) або контракт (F) – логічна змінна, прізвище:

```
> student <- list("ЕП",2,1,T,"Тижненко_О_Г");student
[[1]]
[1] "ЕП"

[[2]]
[1] 2

[[3]]
[1] 1

[[4]]
[1] TRUE

[[5]]
[1] "Тижненко_О_Г".
```

До компонент списку слід звертатися за допомогою подвійних дужок: [[]]. Наприклад,

```
[[5]]
[1] "Тижненко_О_Г"

> student[[1]]
[1] "ЕП"
> student[[5]]
[1] "Тижненко_О_Г"
> student[[2]]
[1] 2.
```

Слід зауважити, що числові компоненти списку припускають звичайні арифметичні дії та функціональні перетворення, наприклад:

```
[1] 2
> student[[2]]/2
[1] 1.
```

Перевірка об'єкта на приналежність до списку здійснюється командою **is.list()**, а перевод об'єкта в список – за допомогою команди **as.list()**.

2.1.2. "data.frame"

Основною формою збереження даних в R є "data.frame", оскільки в економічних дослідженнях усі дані мають назви і можуть бути як числовими, так і текстовими.

Структура "data.frame" складається тільки з векторів, які мають назву. Звертаються до конкретних векторів за їх назвою.

Розглянемо гіпотетичне дослідження 12 біологічних об'єктів, які розрізняються товщиною (Thickness), вагою (Weight), довжиною (Length) та полом (Sex). Порядковий номер об'єкта визначимо змінною Number.

У вікні редактора набираємо:

```
Number <- 1:12
Thickness <- c(1,1,1,2,2,2,3,3,3,4,4,4)
Sex <- c("M","M","F","F","F","M","F","F","F","F","M","F")
Sex <- as.factor(Sex) # дозволяє розглядати дані для "M" і "F" окремо
Weight <- c(2.5, 3.6, 3.8, 4.1, 3.7, 2.8, 3.2, 2.4, 2.9, 3.7, NA, 3.4)
Length <- c(30,33,33,35,34,31,33,30,34,38,32,36)
Lab_1 <- data.frame(Number,Thickness,Sex,Weight,Length)
```

Усе відмічаємо та натискуємо Ctrl+Enter для виконання програми. У вікні Виконання (Console) отримуємо:

```
> Lab_1
   Number Thickness Sex Weight Length
1       1         1   M    2.5     30
2       2         1   M    3.6     33
3       3         1   F    3.8     33
4       4         2   F    4.1     35
5       5         2   F    3.7     34
6       6         2   M    2.8     31
7       7         3   F    3.2     33
8       8         3   F    2.4     30
9       9         3   F    2.9     34
10     10         4   F    3.7     38
11     11         4   M     NA     32
12     12         4   F    3.4     36.
```

Одночасно у вікні Оточення (Environment) з'являється змінна Lab_1. Це означає, що файл даних Lab_1 доступний до обчислень. Він буде знаходитися в Оточенні у продовж сесії R. Якщо є потреба закрити сесію, з'явиться пропозиція зберегти Оточення: "save" або "don't save". Якщо Оточення зберігається, то змінна Lab_1 буде знаходитися у Оточенні, доки користувач не вилучить її за допомогою команди **rm(Lab_1)**.

Доступ до векторів даних здійснюється наступним чином:

```
> Lab_1$Thickness
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
> Lab_1$Weight
 [1] 2.5 3.6 3.8 4.1 3.7 2.8 3.2 2.4 2.9 3.7  NA 3.4
> Lab_1$Sex
 [1] M M F F F M F F F F M F
Levels: F M
```

Доступ до компонент векторів здійснюється звичайним чином:

```
> Lab_1$Weight[3] # третя компонента вектора Lab_1$Weight
[1] 3.8.
```

Зауважимо, що ранні версії R під час складання data.frame усі змінні перетворювали на текстові. Це означало, що неможна було напряму опрацьовувати числову інформацію з data.frame, її приходилося трансформувати до числового типу. Якщо це відбудеться у вашій програмі R, це означає, що слід оновити версію R.

2.2. Збереження даних в R

Під час виконання економічних досліджень не слід інформацію зберігати у Оточенні. По-перше, це зменшує оперативну пам'ять, по-друге, неможлива зручна класифікація даних а потрете дані можуть бути просто загублені.

Для збереження даних на накопичувач існує функція **write.table()**. Заздалегідь необхідно зробити папку, де ці дані будуть зберігатися, наприклад, RTxt, оскільки дані в R зберігаються як текстові файли.

Припустимо, нам необхідно зберегти data.frame, який ми створили, Lab_1. У вікні Виконання (Console) виконуємо команду:

```
> write.table(Lab_1, "d:/RTxt/Lab_1.txt", sep = ","),
```

якщо нам зручно використовувати роздільники у вигляді коми, та

```
> write.table(Lab_1, "d:/RTxt/Lab_1.txt", sep = ""),
```

якщо нам зручно не використовувати роздільники взагалі. У другому варіанті, наприклад, в D:/RText з'явиться файл Lab_1.txt:

"Number""Thickness""Sex""Weight""Length"
"1"11"M"2.530
"2"21"M"3.633
"3"31"F"3.833
"4"42"F"4.135
"5"52"F"3.734
"6"62"M"2.831
"7"73"F"3.233
"8"83"F"2.430
"9"93"F"2.934
"10"104"F"3.738
"11"114"M"NA32
"12"124"F"3.436.

Якщо ми вже убрали ці дані з Оточення, а нам треба з ними щось робити, потрібно ці дані знову завантажити в Оточення. Для цього застосовується функція **read.table()**:

```
Lab_1 <- read.table("D:/RTxt/Lab_1.txt",
            header = TRUE,sep="",dec=".").
```

Якщо ми завантажуємо дані в R, то назву даних можна змінити.

2.3. Імпорт даних

Економічні дані, як, правило зберігаються в Excel і мають назви для кожного показника. Тому в R передбачено, що дані, які імпортуються обов'язково мають назви колонок і повинні бути трансформовані в структуру "data.frame".

Зауважимо, що дані у вигляді структури "data.frame" знаходяться тільки в всередині програмного середовища R, а саме, в його Оточенні. Для того, щоб ввести дані в Оточення, в R існує функція **read.table()**, як було показано у попередньому пункті. Ця функція має такі поля, що необхідно заповнити:

```
Назва_текстового_файлу <- read.table("повний шлях до файлу",
            header = TRUE,sep="",dec=".").
```

Зауважимо також, що опція "header = TRUE" повідомляє R, що колонки даних мають назву. Опція «sep=""» повідомляє, що у текстовому файлі немає роздільників між сусідніми за рядком даними. Замітимо, що дані у рядку можуть розділятися також знаком ";". Тоді цю опцію слід змінити на «sep=";"». Остання необхідна опція це «dec=".»». Цю опцію слід застосовувати, якщо у текстовому файлі десятинним роздільником є точка. Якщо у текстовому файлі десятинним роздільником є точка з комою, то цю опцію слід змінити на «dec =","».

Тобто взагалі, імпорт даних із Excel в R в трансформації даних з формату Excel до певного текстового формату. Зауважимо, що, як правило, але не завжди, дані в Excel мають десятинним роздільником кому.

Підсумовуючи, можна сказати, що імпорт даних з Excel до R складається з двох етапів: з трансформування даних з формату Excel до текстового формату та з трансформування даних з текстового формату до внутрішнього формату R.

Розглянемо трансформування даних із формату Excel до текстового формату. Найпростіший варіант – це зберегти файл Excel як новий файл (наприклад, TestN.xlsx) без зайвих написів, відмітити усі дані (Ctrl + A), скопіювати (Ctrl + C) та вставити (Ctrl + V) в Блокнот, зберегти як текстовий файл (TestN.txt) у спеціальну папку, наприклад RTxt. Далі необхідно ввести дані до Оточення:

```
tempN <- read.table("D:/RTxt/TestN.txt",
            header = TRUE,sep="",dec=".").
```

Назва data.frame – tempN, довільна і може співпадати з назвою файлу (TestN). Нагадаємо, що не слід забувати перевіряти, які десятинні роздільники використовує Excel. У даному випадку використовується *точка:* dec=".".

Дуже зручним іншим методом імпорту *невеликих обсягів даних* з Excel безпосередньо до R є використання оперативної пам'яті комп'ютера (Clipboard). Для цього достатньо використати функцію **read.table()**:

```
# For import data from Clipboard
назва_файлу_латиною <- read.table(file = "clipboard",
                sep = "\t",
                header = T,
                dec = ".",
                stringsAsFactors = T).
```

Для імпорту з Excel необхідно на аркуші Excel скопіювати дані з назвами (Ctrl + C), перейти до R та використати приведену вище функцію **read.table()**. Необхідно тільки скоректувати її, якщо в Excel використовується десятинний роздільник у вигляді коми. Тоді, замість «dec = "."» необхідно написати «dec = ","».

У результаті цієї операції в Оточенні з'являться нові дані: "назва_файлу_латиною", які мають структуру "data.frame". Із цими даними вже можна працювати в R. Однак, щоб завершити процес імпорту даних, необхідно зберегти ці дані у вигляді текстового файлу за допомогою функції **write.table()**, як це показано в пункті 2.2.

Розділ 3. Лінійна регресія

3.1. Основні положення задачі лінійної регресії

3.1.1. Позначення та скорочення

Генеральна сукупність досліджуваних об'єктів: ГС.

Метод найменших квадратів: МНК.

Пакети прикладних програм: ППП.

Випадкова величина: ВВ.

Середнє значення випадкової величини X : $M(X)$ або m_X .

Число степенів свободи ВВ: ЧСС.

Сума квадратів відхилень: $SS_x = \sum_i^n (x_i - \overline{x})^2$.

Матриця спостережених значень регресорів:
$X = (X_j)_{j=1:m} = (X_{ij})_{i=1:n;\, j=1:m}$.

Матриці-стовпчик (вектор) спостережених значень відгуку: Y .

Коефіцієнт регресії при X_j лінійної моделі в натуральних змінних у ГС: b_j .

Вибіркова оцінка коефіцієнта регресії при X_j в лінійної моделі у натуральних змінних: $\hat{b}_j$.

Теоретичне стандартне відхилення коефіцієнта регресії b_j : σ_{b_j} .

Оцінка стандартного відхилення вибіркового коефіцієнта регресії ($\hat{b}_j$) при X_j на основі відповідної теоретичної формули: $s_{\hat{b}_j}$.

Залишкова помилка лінійної регресійної моделі у ГС в натуральних змінних: ε;

Стандартне відхилення залишкової помилки у ГС: σ_ε.

Залишкова помилка вибіркової лінійної регресійної моделі в натуральних змінних: e;

Стандартне відхилення вибіркової залишкової помилки : s_e.

Матриця спостережених значень регресорів X у стандартизованій формі: $T = (t_j)_{j=1:m} = (t_{ij})_{i=1:n;\, j=1:m}$.

Матриці-стовпчик (вектор) спостережених значень відгуку Y у стандартизованій формі: t_Y.

Коефіцієнт регресії при t_j лінійної моделі в стандартизованих змінних у ГС: β_j.

Оцінка коефіцієнта регресії при t_j лінійної моделі в стандартизованих змінних: $\hat{\beta}_j$.

Помилка I-го роду: α.

Помилка II-го роду: β.

p-value: p_v.

Потужність критерію: p_w.

3.1.2. Основні форми запису лінійної регресійної моделі

У практиці економічних досліджень використовуються дві форми моделі лінійної регресії: модель у натуральних змінних та модель у стандартизованих змінних.

Слід звернути увагу на те, що усі пакети прикладних програм використовують МНК тільки для моделі регресії у натуральних змінних оскільки теоретична формула розрахунку дисперсій вибіркових коефіцієнтів регресії для випадку нормальних нестохастичних регресорів існує тільки для регресії у натуральних змінних

З іншого боку, при наявності мультиколінеарності на сьогодення нема іншої альтернативи, як використати ridge-регресію замість звичайного МНК. Однак Ridge-регресія може бути застосована, як правило, для моделі у стандартизованих змінних.

Тому, перш ніж вирішувати задачу регресії, необхідно провести діагностику на мультиколінеарність. Якщо мультиколінеарності нема, можна застосовувати звичайний МНК у будь-якому пакеті, наприклад, у найбільш поширеному у економічних дослідженнях, R, це може бути програма **lm.R** у пакеті {Stat}. Якщо мультиколінеарність є, то необхідно знайти програму, яка використовує модель ridge-регресії у стандартизованої формі, а саме – програму **lm.ridge.R** у пакеті {MASS}.

3.2. Лінійна регресійна модель у натуральних змінних

Натуральними змінними є спостережені дані: відгук *Y*, та матриця регресорів *X*, які створюють структуру data.frame в R. Статистичний зв'язок між відгуком та *m* регресорами у ГС для лінійної моделі має вигляд векторного рівняння, яке з математичної точки зору є наближеним розкладом вектора відгуку Y за векторами регресорів X_j:

$$Y = b_0 + \sum_{j=1}^{m} b_j X_j + \varepsilon . \qquad (1)$$

Зауважимо, що крім спостережених даних (Y, X) в моделі (1) фігурують неспостережені величини (b_0, b_j, ε), які ми не знаємо і не можемо визначити точно, але котрі являють собою саме ті величини, які тільки й цікавлять економістів: (b_0, b_j), значення коефіцієнтів регресії у ГС, та ε, величина залишкової помилки у ГС.

Слід також зауважити, що в рівнянні (Y, X) є спостережені дані, які ми вибрали випадковим чином з ГС і які являють собою вибірку розміру *n*. Тому, з теоретичної точки зору потрібні нам значення коефіцієнтів регресії у ГС, тобто (b_0, b_j), можна було б знайти з рівняння (1), якщо б (Y, X) являли собою усі дані ГС, що абстрактно можна відобразити у вигляді $n \to \infty$. Хоча ясно, що реально розмір вибірки (n) не може бути нескінченним, ця ідея має реальний та дуже важливий сенс для економіста: зі зростанням розміру вибірки її статистичні характеристики (тобто вибіркові оцінки характеристик у ГС) повинні наближатися до їх значень у ГС, що називається у статистиці *спроможністю* оцінок.

Якщо матричне рівняння (1) вирішується для конкретних спостережених даних ($Y=(Y_i)_{i=1:n}$, $X=(X_{ij})_{i=1:n;\,j=1:m}$), тобто не всіх даних ГС, то рішенням рівняння (1) є оцінки, $\hat{b}_0$ та $\hat{b}_j$, коефіцієнтів регресії у ГС, тобто величин (b_0, b_j), точне значення яких ми визначити не можемо. Якщо ми проведемо ще один статистичний експеримент на інших об'єктах тієї самої ГС, то отримуємо іншу вибірку даних ($Y=(Y_i)_{i=1:n}$, $X=(X_{ij})_{i=1:n;\,j=1:m}$), тобто іншу реалізацію випадкових величин (Y, X), то вирішуючи рівняння (1), отримуємо інші оцінки $\hat{b}_0$ та $\hat{b}_j$. Тобто, оцінки $\hat{b}_0$ та $\hat{b}_j$ змінюються від реалізації до реалізації.

Виникає практичне питання, наскільки значно оцінки $\hat{b}_0$ та $\hat{b}_j$, тобто рішення рівняння (1), змінюються від реалізації до реалізації. Апріорі зрозуміло, що у тому випадку, коли зміна оцінок $\hat{b}_0$ та $\hat{b}_j$ від реалізації до реалізації має порядок самих оцінок, які ми отримуємо у результаті рішення рівняння (1), то знаходження оцінок $\hat{b}_0$ та $\hat{b}_j$ з одного рівняння (1), тобто для одної вибірки, дуже неточно оцінює коефіцієнти регресії у ГС. Оскільки розкид значень оцінок $\hat{b}_0$ та $\hat{b}_j$ характеризує їх дисперсія, то *основною проблемою регресійного аналізу* і є знаходження стандартних відхилень величин $\hat{b}_0$ та $\hat{b}_j$ від реалізації до реалізації, тобто величин σ_{b_0} та σ_{b_j}.

Проблема знаходження величин σ_{b_0} та σ_{b_j} ускладнюється тим, що хоча ці величини знаходяться теоретично у випадку нормально розподілених даних та нестохастичних регресорів, але вони залежать від стандартного відхилення залишкової помилки σ_ε, яка невідома. Замість цієї величини беруть її оцінку, стандартне відхилення вибіркової залишкової помилки, тобто величину s_e, котра знаходиться з вибіркового рівняння регресії, тобто рівняння (1) для вибіркових даних.

Для рішення задачі регресії, тобто знаходження коефіцієнтів регресії $\hat{b}_0$ та $\hat{b}_j$ за даними вибірки, у всіх ППП векторне рівняння (1) перетворюють до матричної форми:

$$Y = Db + \varepsilon, \qquad (2)$$

де матриця $D = (1, X)$ складається з колонки одиниць розміру *n* та матриці регресорів X і називається "*дизайн-матриця*" (design matrix). На цей факт слід звернути особливу увагу під час використання R для розв'язання задачі лінійної регресії, оскільки у коментарях до відповідних функцій дизайн-матриця позначається так точно, як і матриця спостережених регресорів, тобто X .

Метод найменших квадратів (МНК) розв'язання рівняння (2) практично міститься у наступному. Рівняння (2) множиться на транспоновану дизайн-матрицю D' та використовується так звана *система нормальних рівнянь*:

$$D'\varepsilon = 0, \tag{3}$$

після чого залишається рівняння:

$$D'Db = D'Y. \tag{4}$$

Матричне рівняння (4) є звичайна визначена система лінійних рівнянь розміру [*m*+1,*m*+1], оскільки ми додали до *m* регресорів ще один вектор одиниць. Це рівняння можна вирішувати будь-яким способом, який є в R. Слід мати на увазі, що у рівнянні (4) невідомі $b = (b_0, b_1, ..., b_m)$ позначаються як $\hat{b} = (\hat{b}_0, \hat{b}_1, ..., \hat{b}_m)$ після знаходження їх вибіркових значень.

Стандартною функцією рішення систем лінійних рівнянь в R є функція **solve(*A*,*B*)**, де *A* – матриця системи, *B* – вектор правої частини. Тобто рішення рівняння (4) можна знайти в R як:

$$\hat{b} = solve(D'D, D'Y). \tag{5}$$

Приклад 1

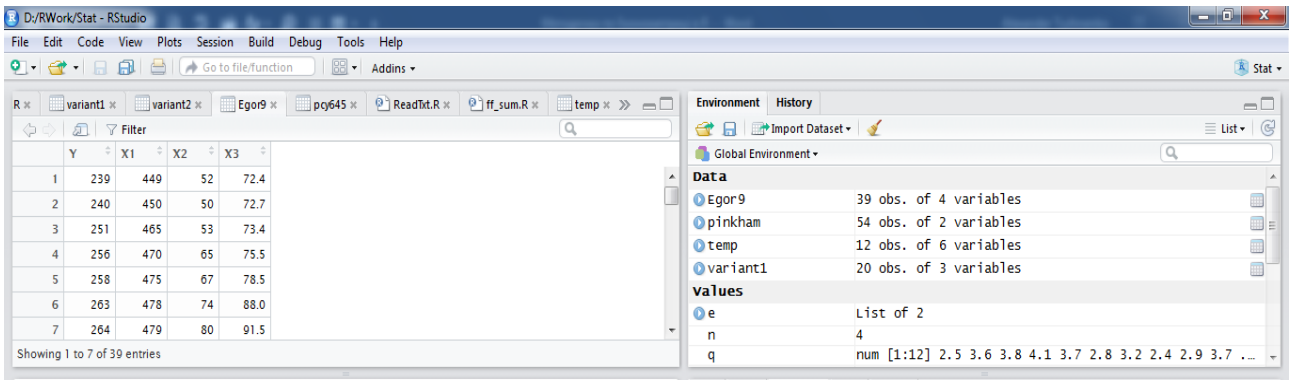
Спостережувані дані:
Y X1 X2 X3
239 449 52 72.4
240 450 50 72.7
251 465 53 73.4
256 470 65 75.5

258 475 67 78.5
263 478 74 88.0
264 479 80 91.5
267 482 85 93.0
270 485 89 95.5
272 487 97 98.8
274 492 99 105.8
275 493 108 107.2
276 494 111 107.9
278 498 117 114.8
283 502 128 122.7
286 504 134 134.6
290 509 137 147.5
291 512 144 150.3
293 520 153 170.7
295 525 168 192.1
298 528 172 194.6
300 532 173 198.8
302 537 175 202.7
304 539 178 206.1
305 541 181 208.8
308 546 185 216.6
311 549 189 220.7
318 554 193 229.0
319 557 205 239.0
322 560 215 254.6
325 564 224 267.9
327 569 231 275.0
328 573 238 275.4
332 574 244 290.5
335 580 248 295.7
338 584 256 311.2
340 590 265 322.9
344 598 274 336.5
346 605 281 346.8

набрати у Блокноті Windows, зберегти як Egor9.txt у папці \RTxt. У редакторі R набрати програму як новий скрипт:

```
Egor9 <- read.table("D:/RTxt/Egor9.txt",
                    header = TRUE,sep="",dec="."),
```

та зберегти її з галочкою "Source on save". Новий data.frame з назвою Egor9 з'явиться у Глобальному Оточенні, а також у списку редактора:



Визвати, наприклад, відгук можна наступним чином:

```
Egor9$Y
 [1] 239 240 251 256 258 263 264 267 270 272 274 275 276 278 283 286 290 291
293
[20] 295 298 300 302 304 305 308 311 318 319 322 325 327 328 332 335 338 340
344
[39] 346
```

Можна далі оперувати вектором Egor9$Y, а можна означити його зручніше, як він був означений:

```
> Y <- Egor9$Y
> Y
 [1] 239 240 251 256 258 263 264 267 270 272 274 275 276 278 283 286 290 291
293
[20] 295 298 300 302 304 305 308 311 318 319 322 325 327 328 332 335 338 340
344
[39] 346
```

Аналогічно:

```
> X1 <- Egor9$X1
> X2 <- Egor9$X2
> X3 <- Egor9$X3.
  X0 <- rep(1,n)
```

Визначаємо розмір вибірки:

```
> n <- length(Y);n
[1] 39.
```

Створюємо вектор з одиниць:

```
> X0 <- rep(1,n);X0
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1.
```

Створюємо дизайн-матрицю *D*:

```
> D=matrix(,nrow=n,ncol=4)
# Заповнюємо її векторами X0, X1, X3, X3:
> D[,1] <- X0
> D[,2] <- X1
> D[,3] <- X2
> D[,4] <- X3
# Отримуємо:
> D
      [,1] [,2] [,3]  [,4]
  [1,]    1  449   52  72.4
  [2,]    1  450   50  72.7
  [3,]    1  465   53  73.4
  [4,]    1  470   65  75.5
  [5,]    1  475   67  78.5
  [6,]    1  478   74  88.0
  [7,]    1  479   80  91.5
  [8,]    1  482   85  93.0
  [9,]    1  485   89  95.5
[10,]    1  487   97  98.8
[11,]    1  492   99 105.8
[12,]    1  493  108 107.2
[13,]    1  494  111 107.9
[14,]    1  498  117 114.8
[15,]    1  502  128 122.7
[16,]    1  504  134 134.6
[17,]    1  509  137 147.5
[18,]    1  512  144 150.3
[19,]    1  520  153 170.7
[20,]    1  525  168 192.1
[21,]    1  528  172 194.6
[22,]    1  532  173 198.8
[23,]    1  537  175 202.7
[24,]    1  539  178 206.1
[25,]    1  541  181 208.8
[26,]    1  546  185 216.6
[27,]    1  549  189 220.7
[28,]    1  554  193 229.0
```

```
[29,]    1  557  205 239.0
[30,]    1  560  215 254.6
[31,]    1  564  224 267.9
[32,]    1  569  231 275.0
[33,]    1  573  238 275.4
[34,]    1  574  244 290.5
[35,]    1  580  248 295.7
[36,]    1  584  256 311.2
[37,]    1  590  265 322.9
[38,]    1  598  274 336.5
[39,]    1  605  281 346.8.
```

Для рішення системи (4) складемо матрицю $A = D'D$ та праву частину $B = D'Y$:

```
> A <- t(D)%*%D;A # Матриця МНК-системи (4):
       [,1]     [,2]    [,3]      [,4]
[1,]    39.0    20449    6138    7085.7
[2,] 20449.0 10792425 3330723 3854057.8
[3,]  6138.0  3330723 1147158 1338431.9
[4,]  7085.7  3854058 1338432 1566732.1

> B <- t(D)%*%Y  # Права частина МНК-системи (4):
> B
       [,1]
[1,]   11523
[2,] 6090662
[3,] 1891587
[4,] 2189255.
```

Рішення МНК-системи (4), тобто шукані оцінки коефіцієнтів регресії:

```
> b <- solve(A,B) # Функція, яка вирішує рівняння: Ab=B.
> b
           [,1]
[1,] -47.8164944
[2,]   0.6409222
[3,]   0.2321520
[4,]  -0.1613565
```

Таким чином, оцінки коефіцієнтів регресії дорівнюють: $b_0 = -47.8164944$; $b_1 = 0.6409222$; $b_2 = 0.2321520$; $b_3 = -0.1613565$.

Як видно з наведених даних, відгук Y та усі регресори зростають зі збільшенням індексу. Якщо ми згадаємо загальноприйнятне означення економічного змісту коефіцієнтів регресії, а саме: значення коефіцієнту регресії $b_j > 0$ ($b_j < 0$) визначає величину приросту (зменшення) відгуку

в разі збільшення регресора X_j на одиницю, якщо усі інші регресори не змінюються, то становиться ясно, що при усіх змінних, які одночасно зростають за індексом, усі коефіцієнти регресії повинні бути додатними!

У рішенні, яке ми знайшли, третій коефіцієнт регресії від'ємний ($b_3 = -0.1613565$). Це означає, що з економічній точці зору, рішення не є коректним. Некоректний результат у даному разі пов'язаний із так званою *мультиколінеарністю*, яка буде розглянута далі у зв'язку з ridge-регресією.

3.2.1. Коефіцієнт детермінації

Коефіцієнт детермінації позначається звичайно як R^2 і відіграє важливу роль в економічної інтерпретації результатів рішення задачі лінійної регресії – він показує наскільки якісно регресійна модель відображає відгук. R^2 є величиною додатною та змінюється від 0 до 1. Вважається, що коефіцієнт детермінації повинен перевищувати 0.8 для того, щоб регресія вважалася прийнятною.

Коефіцієнт детермінації визначається з рівняння дисперсій для вибіркового рівняння регресії:

$$Y = \hat{Y} + e,\ \hat{Y} = \hat{b}_0 + \sum_{j=1}^{m} \hat{b}_j X_j\,, \tag{6}$$

у якому відгук Y є представленим як розрахункова невипадкова функція $\hat{Y}$ плюс квазівипадкова залишкова помилка e. Величина e називається квазівипадковою тому, що на неї накладена умова:

$$\overline{e} = 0 \Leftrightarrow \frac{1}{n}\sum_{i=1}^{n} e_i = 0 \Rightarrow \sum_{i=1}^{n} e_i = 0\,. \tag{7}$$

Слід зауважити, що у тому випадку, коли система нормальних рівнянь МНК записана у вигляді (3), у вибірковому рівнянні регресії (6) вже нічого визначати, бо оцінки регресій $(\hat{b}_0, \hat{b}_j)$ вже знайдені. У цьому випадку

рівняння (6) є просто представлення відгуку Y у вигляді розрахункової функції $\hat{Y}$ та квазівипадкової помилки e, яка є відомою та дорівнює:

$$e = Y - \hat{Y} = Y - \hat{b}_0 - \sum_{j=1}^{m} \hat{b}_j X_j . \quad (8)$$

Примітка. Треба знати, що є й інший підхід до визначення оцінок коефіцієнтів регресії лінійної моделі, коли не застосовують систему нормальних із самого початку, а починають з вибіркового рівняння регресії у вигляді:

$$Y = b_0 + \sum_{j=1}^{m} b_j X_j + e , \quad (9)$$

де вважається, що величини b_0, b_j та e невідомі. У цьому разі оцінки коефіцієнтів регресії знаходяться з умови мінімізації середньоквадратичної помилки (МНК), тобто функції:

$$L(b_0, b_1, ..., b_m) = \sum_{i=1}^{n} \left(Y - b_0 - \sum_{j=1}^{m} b_j X_j \right) \to min , \quad (10)$$

для якої існує тільки один глобальний мінімум у точках $(\hat{b}_0, \hat{b}_1, ..., \hat{b}_m)$, які ми знайшли більш простим методом, використовуючи систему нормальних рівнянь (3), яка витікає з умови (10).

Відповідно рівнянню (6) складається рівняння невиправлених дисперсій [Greene, p.81]:

$$\frac{1}{n} SS_Y = \frac{1}{n} SS_{\hat{Y}} + \frac{1}{n} SS_e , \quad (11)$$

з якого отримуємо коефіцієнт детермінації:

$$R^2 = \frac{SS_{\hat{Y}}}{SS_Y} , \quad (12)$$

а також формулу суми квадратів залишкової помилки:

$$SS_e = SS_Y(1-R^2), \tag{13}$$

знання якої необхідно для оцінки значимості вибіркових коефіцієнтів кореляції.

Приклад 2

Знайти коефіцієнт детермінації для задачі регресії, яка розв'язана в прикладі 1.

Для цього, згідно з формулою (12), необхідно знайти значення розрахункової функції

$$\hat{Y}_i = \hat{Y}(X_{ij}) = \hat{b}_0 + \sum_{j=1}^{m} \hat{b}_j X_{ij}$$

та знайти $SS_{\hat{Y}} = \sum_{i=1}^{n} (\hat{Y}_i - M(\hat{Y}_i))^2$, де $M(\hat{Y}_i)$ є середнє значення змінної $\hat{Y}$.

Рішення.

```
SSyc <- b[1]+b[2]*X1+b[3]*X2+b[4]*X3
# Розрахункова функція
> SSYc <- var(SSyc)
# Дисперсія розрахункової функції
> SSY <- var(Y)
# Дисперсія відгуку
> R2 <- SSYc/SSY # Коефіцієнт детермінації
> R2
[1] 0.9967449.
```

Тут використаний той факт, що дисперсія випадкової величини X в R обчислюється за формулою:

$$\mathrm{var}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 = \frac{SS_X}{n-1},$$

тому

$$\frac{SS_{\hat{Y}}}{SS_Y} = \frac{\mathrm{var}(\hat{Y})}{\mathrm{var}(Y)},$$

що суттєво спрощує програму обчислення суми квадратів випадкових величин за рахунок використання вбудованих функцій R.

Відповідь: Коефіцієнт детермінації R^2 = 0.997.

3.2.2. Перевірка на значимість регресійної моделі

Значимість або незначимість лінійної регресійної моделі міститься у тому наскільки вибіркова модель, тобто її параметри (коефіцієнти регресії $(\hat{b}_0, \hat{b}_j)$), близька або не близька до моделі у ГС, тобто, коефіцієнтів регресії (b_0, b_j).

Оскільки параметри моделі у ГС (b_0, b_j) взагалі невідомі, питання значимості моделі розглядається теоретично у припущенні нормального розподілу усіх змінних та нестохастичності регресорів.

Нестохастичність регресорів не означає, що регресори не є випадковими, але припускає, що від реалізації до реалізації (від вибірки до вибірки) змінюється тільки відгук, а регресори залишаються незмінними.

Для перевірки *m*-факторної лінійної моделі на значимість використовують критерій Фішера, який порівнює спостережену величину додатної статистики Фішера:

$$F = \frac{S^2_{\hat{Y}}}{S^2_e}, \tag{14}$$

(де $S^2_{\hat{Y}}$ – виправлена вибіркова дисперсія регресії; S^2_e – виправлена вибіркова дисперсія залишкової помилки) з критичним значенням цієї статистики $F_{\alpha,m,n-m-1}$ на рівні α (0.05 або 0.01). Водночас формулюються дві гіпотези:

$$\begin{cases} H_0 : усі\ b_j\ дорівнюють\ нулю \\ H_a : не\ всі\ b_j\ дорівнюють\ нулю \end{cases}. \tag{15}$$

Якщо усі змінні нормальні, тобто є розподіленими за нормальним законом, то статистика Фішера (14) є розподіленою за законом χ^2 з $n-m-1$ числом степенів свободи.

Слід також зауважити, що статистика (14) є додатною, тому для з'ясування факту справедливості гіпотези слід використовувати однобічний тест. Цей факт відображається у тому, що критичне значення статистики $F_{\alpha,m,n-m-1}$ на рівні α залежить саме від α, а не від $\alpha/2$, як було б при двобічному тесті.

Для обчислення спостереженого значення статистики Фішера слід використати наступну формулу, яка виходить із формули (14):

$$F = \frac{n-m-1}{m}\frac{R^2}{1-R^2}. \qquad (16)$$

Для обчислення критичного значення статистики $F_{\alpha,n-m-1}$ в R, можна використати квантиль-функцію розподілу Фішера **qf**(p, df1, df2), де p – ймовірність, а df1 **–** число степенів свободи розрахункової функції, яке дорівнює числу факторів *m*; df2 **–** число степенів свободи залишкової помилки, яке дорівнює *n*-*m*-1.

Якщо спостережене значення статистики $F < F_{\alpha,m,n-m-1}$, то ми не маємо підстави відхилити нульову гіпотезу. У цьому випадку модель вважається незначимою.

Якщо спостережене значення статистики $F > F_{\alpha,m,n-m-1}$, то ми маємо підстави відхилити нульову гіпотезу. У цьому випадку модель вважається значимою.

Приклад 3

Перевірити на значимість лінійну модель у прикладі 1 за критерієм Фішера на рівні 0.05.

Рішення. Розраховуємо спостережене значення статистики Фішера, використовуючи вже знайдений у прикладі 2 коефіцієнт детермінації, R^2 = 0.997:

```
[1] 3.940299
> F <- ((n-4)/4)*R2/(1-R2)
> F
[1] 2679.301.
```

Знаходимо далі критичне значення статистики $F_{\alpha,m,n-m-1}$, у даному разі це $F_{0.05,3,35}$:

```
> qf(.95,3,35)
[1] 2.874187
```

Тобто: $F_{0.05,3,35}$ = 2.874187. Висновок: оскільки спостережене значення статистики Фішера (2679.301) перевищує критичне значення статистики $F_{0.05,3,35}$ (2.874187), то модель значима на рівні 0.05.

3.2.3. Перевірка коефіцієнтів регресії на значимість за Стьюдентом

Якщо перевірка значимості лінійної моделі за Фішером показала, що модель не значима, то перевіряти коефіцієнти регресії на значимість не потрібно. Просто у цьому випадку лінійна модель не може бути використаною для економічного аналізу.

Якщо ж модель значима за Фішером, то згідно до визначення альтернативної гіпотези (15), хоча б один з коефіцієнтів регресії у ГС не дорівнює нулю. Коефіцієнт регресії у ГС не дорівнює нулю, якщо він є значимим за критерієм Стьюдента. Тобто у цьому випадку необхідно перевірити усі коефіцієнти регресії на значимість за Стьюдентом, та з'ясувати, саме які коефіцієнти регресії у ГС не дорівнюють нулю.

Перевірити усі коефіцієнти регресії на значимість за Стьюдентом можна за допомогою *t*-тесту. Для цього для кожного вибіркового коефіцієнта регресії $\hat{b}_j$, крім $\hat{b}_0$, необхідно розрахувати значення *t*-статистики:

$$t = \frac{\hat{b}_j}{s_{\hat{b}_j}} \tag{17}$$

та порівняти це значення з критичним значенням розподілу Стьюдента із числом степенів свободи $df = n - 2$, де *n* є розміром вибірки, оскільки величина (17) розподілена за законом Стьюдента саме з $df = n - 2$, від вибірки до вибірки – тобто в разі багаторазового проведення економічного експерименту.

Для визначення спостереженого значення статистики Стьюдента необхідно знати крім вибіркових коефіцієнтів регресії $\hat{b}_j$, ще й їх стандартне відхилення $s_{\hat{b}_j}$, яке розраховується за наступною формулою:

$$s_{\hat{b}_j} = \sqrt{\sigma_\varepsilon^2 diag((D'D)^{-1})_j}, \quad (18)$$

де σ_ε^2 є дисперсія залишкової помилки у ГС а $diag((D'D)^{-1})_j$ є *j*-й елемент головної діагоналі оберненої матриці $(D'D)^{-1}$, а D є дизайн-матриця даних. Величина σ_ε^2 нам не відома, тому її замінюють на оцінку:

$$\sigma_\varepsilon^2 \approx \frac{SS_e}{n-m-1},$$

де SS_e визначається формулою (13). У результаті ми отримуємо наступну оцінку для σ_ε^2:

$$\sigma_\varepsilon^2 \approx \frac{SS_Y(1-R^2)}{n-m-1} \quad (19)$$

та відповідно оцінку стандартного відхилення $s_{\hat{b}_j}$:

$$s_{\hat{b}_j} = \sqrt{\frac{SS_Y(1-R^2)}{n-m-1} diag((D'D)^{-1})_j}, \quad (20)$$

яка дає можливість розрахувати *t*-статистику (17) для кожного коефіцієнта регресії.

Для проведення *t*-тесту, для кожного коефіцієнту регресії $\hat{b}_j$ необхідно сформулювати нульову та альтернативну гіпотези:

$$\begin{cases} H_0 : b_j = 0 \\ H_a : b_j \neq 0 \end{cases}. \quad (21)$$

Це формулювання означає, що згідно з гіпотезою H_0, коефіцієнт регресії b_j у ГС дорівнює нулю, а те, що у результаті експерименту ми отримали $\hat{b}_j \neq 0$ є випадкова подія, математичне сподівання якої дорівнює нулю. У той же час, згідно з гіпотезою H_a, коефіцієнт регресії b_j у ГС не дорівнює нулю, тобто математичне сподівання отриманої в результаті експерименту величини $\hat{b}_j$ не дорівнює нулю.

Зміст *t*-тесту міститься у наступному: прийняти чи ні гіпотезу H_0. Відносно гіпотези H_a *t*-тест не дає можливості визначитися з прийняттям або відхилення цієї гіпотези, але формулювання гіпотези H_a суттєво впливає на форму *t*-тесту: якщо $H_a : b_j \neq 0$, то застосовується двобічний *t*-тест, якщо $H_a : b_j > 0$, то застосовується однобічний *t*-тест.

Якщо перевіряється гіпотеза $H_0 : b_j = 0$ для коефіцієнта регресії, то передбачається, що цей коефіцієнт може бути, як більше нуля, так і менше нуля – у такому разі використовується двобічний *t*-тест, оскільки ця статистика має знак коефіцієнта регресії.

Якщо перевіряється лінійна модель за критерієм Фішера, наприклад, то статистика Фішера (16) не може бути від'ємною взагалі, тому ми застосовували однобічний тест Фішера.

Перевірка H_0 у формулюванні (21) на рівні α відбувається за допомогою визначення $(1-\alpha)100\%$ довірчого інтервалу для *t*-статистики (17). Якщо H_0 є вірною, то зі ймовірністю $(1-\alpha)$ *t*-статистика (17) повинна знаходитися в наступному інтервалі:

$$-t_{1-\alpha/2,n-2} < t < t_{1-\alpha/2,n-2}. \qquad (22)$$

Якщо, після обчислення *t*-статистики (17), з'ясовано, що вона дійсно знаходиться у цьому інтервалі, то приймається гіпотеза H_0 та формулюється інференція: коефіцієнт регресії b_j ***є незначимим*** на рівні α.

Якщо з'ясовано, що статистика не знаходиться у цьому інтервалі, то гіпотеза H_0 відхиляється та формулюється інференція: коефіцієнт регресії b_j ***є значимим*** на рівні α. Стосовно гіпотези H_0 інференція не формулюється взагалі.

Для розрахунку критичних значень *t*-статистики, $t_{1-\alpha/2,n-2}$, в R можна використати квантиль-функцію *t*-розподілу qt(p,df) для $p=1-\alpha/2$ та $df=n-2$.

Приклад 4

Перевірити на значимість коефіцієнти регресії лінійної моделі, які отримані у прикладі 1 за критерієм Стьюдента на рівні 0.05.

Рішення. Для визначення *t*-статистики використовуємо вже відомі коефіцієнти регресії:

```
> b
          [,1]
[1,] -47.8164944
[2,]   0.6409222
[3,]   0.2321520
[4,]  -0.1613565
```

Знайдемо оцінку теоретичної помилки коефіцієнтів регресії за формулою (20):

$$s_{\hat{b}_j}=\sqrt{\frac{SS_Y(1-R^2)}{n-m-1}diag((D'D)^{-1})_j}\,. \qquad (23)$$

Величину SS_Y простіше знайти R зі стандартної формули **var()**:

```
> SSy <- (n-1)*var(Y)
> SSy
[1] 34043.69.
```

Коефіцієнт детермінації нам вже відомий:

```
> R2
[1] 0.9967449
```

Дизайн-матриця *D* теж відома. Тому знаходимо обернену матрицю $(D'D)^{-1}$. для цього використовуємо функцію **solve()**:

```
> id <- solve(t(D)%*%D)
> id
```

```
             [,1]          [,2]          [,3]          [,4]
[1,] 301.74484708 -0.7052700412  0.3721024072  0.0523665216
[2,]   -0.70527004  0.0016495164 -0.0008807579 -0.0001156324
[3,]    0.37210241 -0.0008807579  0.0008372984 -0.0002315526
[4,]    0.05236652 -0.0001156324 -0.0002315526  0.0002460651
```

Елементи головної діагоналі є $diag((D'D)^{-1})_j$ у формулі (23):

```
> d <- diag(id)
> d
[1] 3.017448e+02 1.649516e-03 8.372984e-04 2.460651e-04.
```

Знаючи вибіркові коефіцієнти регресії та діагональні елементи матриці $(D'D)^{-1}$, знаходимо вектор значень *t*-статистики:

```
> t <- sqrt(SSy*(1-R2)*d/(n-4))
> t # допоміжна змінна
[1] 30.90932603  0.07226831  0.05148844  0.02791223
> t <- b/t
> t # вектор значень t-статистики:
            [,1]
[1,] -1.546992
[2,]  8.868648
[3,]  4.508818
[4,] -5.780853
```

Знайдені значення *t*-статистики слід зіставити з 95 % критичним значенням *t*-розподілу:

```
> qt(.975,37)
[1] 2.026192.
```

Зіставляючи, бачимо, що усі три коефіцієнти регресії, $b_1, b_2 \; i \; b_3$, є значимими на рівні $\alpha = 0.05$, оскільки не належать інтервалу (22), враховуючи, що $t_{1-\alpha/2,n-2}$ = 2.026192 при *n* = 39.

3.3. Більш просунута програма рішення лінійної багатофакторної регресії

Завантажуємо текстовий файл із даними:

```
Egor9 <- read.table("D:/RTxt/Egor9.txt")
```

безпосередньо у Глобальне Оточення (ГО). Кликнув на файлі у ГО слід продивитися назви векторів даних. Після цього зручно записати відгук та регресори скорочено:

```
Y <- Egor9$Y
X1 <- Egor9$X1
X2 <- Egor9$X2
X3 <- Egor9$X3
```

та створити дизайн-матрицю:

```
> X0 <- rep(1,n)
> X<-c(X0,X1,X2,X3)
n <- nrow(X)
p <- ncol(X).
```

Параметри моделі (коефіцієнти кореляції):

```
> b.hat <- solve(t(X)%*%X)%*%t(X)%*%Y
> b.hat
        [,1]
X0 -47.8164944 # b̂0
X1   0.6409222 # b̂1
X2   0.2321520 # b̂2
X3  -0.1613565 # b̂3
```

Вектор регресії (розрахункова функція):

```
> y.hat <- X%*%b.hat.
```

Виправлена дисперсія та стандартне відхилення залишкової помилки (s_e^2, s_e):

```
> se2 <- sum((Y - y.hat)^2)/(n - p)
> se <- sqrt(se2)
> se
[1] 1.779384.
```

Величина $(D'D)^{-1}s_e^2$:

```
> v <- solve(t(X) %*% X) * se2
> v
```

```
                X0              X1              X2              X3
X0  955.3864357  -2.2330304477   1.1781529858   0.1658032106
X1   -2.2330304   0.0052227092  -0.0027886613  -0.0003661161
X2    1.1781530  -0.0027886613   0.0026510595  -0.0007331434
X3    0.1658032  -0.0003661161  -0.0007331434   0.0007790927
```

Стандартне відхилення параметрів моделі:

```
> sb <- sqrt(diag(v))
> sb
          X0           X1           X2           X3
30.90932603  0.07226831  0.05148844  0.02791223
> t.values <- b.hat/sb
> t.values
           [,1]
X0 -1.546992
X1  8.868648
```

Спостережені значення *t*-статистики для кожного параметра:

```
X2  4.508818
X3 -5.780853
```

Величини *p*-value для *t*-тесту кожного параметра

```
> pw <- 2 * (1 - pt(abs(t.values), n - p))
> pw
         [,1]
X0 1.308605e-01
X1 1.779037e-10
X2 7.003550e-05
X3 1.501461e-06.
```

Застосування стандартної функції рішення багатофакторної регресії в R:

```
> ex1 <- lm(Y ~ X1 + X2 + X3, Egor9)
> summary(ex1)

Call:
lm(formula = Y ~ X1 + X2 + X3, data = Egor9)

Residuals:
   Min     1Q  Median     3Q    Max
-3.2177 -1.3312 -0.3244  1.3115  3.5824
```

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-47.81649	30.90933	-1.547	0.131	
X1	0.64092	0.07227	8.869	1.78e-10	***
X2	0.23215	0.05149	4.509	7.00e-05	***
X3	-0.16136	0.02791	-5.781	1.50e-06	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.779 on 35 degrees of freedom
Multiple R-squared: 0.9967, Adjusted R-squared: 0.9965
F-statistic: 3572 on 3 and 35 DF, p-value: < 2.2e-16.

Використана література

1. Crawley M. J. The R Book / M. J. Crawley. – Chichester : Wiley & Sons Ltd, 2007. – 601 p.

2. Chambers John M. Software for Data Analysis. Programming with R / John M. Chambers. – New York : Springer, 2008. – 213 p.

3. R Development Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing. – [Electronic resource]. – Vienna, Austria : s. n., 2009. – 113 p. – Access mode : http://www.R-project.org.

4. Everitt B. S. A Handbook of Statistical Analyses Using R / B. S. Everitt, T. Hothorn. – London ; Erlangen : s. n., 2005. – 342 p.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації до самостійної роботи з математичних дисциплін з використанням програмного середовища R

для студентів усіх спеціальностей першого (бакалаврського) рівня

Самостійне електронне текстове мережеве видання

Укладачі: **Малярець** Людмила Михайлівна
Тижненко Олександр Григорович

Відповідальний за видання *Л. М. Малярець*

Редактор *О. В. Анацька*

Коректор *О. В. Анацька*

План 2017 р. Поз. № 236 ЕВ. Обсяг 85 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру
ДК № 4853 від 20.02.2015 р.